NEDERLANDSTALIG HANDBOEK

sinclair ZX Spectrum



sinclair

ZX SPECTRUM

Handleiding voor de sinclair ZX SPECTRUM

door:

Steven Vickers en
Robin Bradbeer

Oorspronkelijke titels:
Sinclair ZX Spectrum Introduction,
door Steven Vickers en Robin Bradbeer,
uitgegeven door Sinclair Research Ltd.,
Cambridge 1982.
Sinclair ZX Spectrum BASIC programming
door Steven Vickers en Robin Bradbeer,
uitgegeven door Sinclair Research Ltd.,
Cambridge 1982.

Vertaling: Wichert van Engelen
Zetwerk: Wolfkamp/Perscombinatie, Amsterdam
Druk: Drukkerij Oldemarkt B.V., Oldemarkt

ISBN: 90-6854-001-7 (gelijmde uitvoering)

ISBN: 90-6854-002-5 (gebonden uitvoering)

SISO: 365.3

Een uitgave van:

Micropress
Leidsedreef 2
2352 BA Leiderdorp

INTRODUKTIE

Inhoud

1. Het aansluiten van de computer

Dit korte deel van het boek is geschreven voor twee soorten mensen. Allereerst voor diegenen die niets, of bijna niets, van computers weten. Ten tweede voor diegenen die wel gewend zijn aan het gebruik van computersystemen, maar die graag eerst een instructie lezen voor ze alles aansluiten.
Het tweede deel van het boek is de BASIC-programmeerhandleiding. De beginner moet eerst dit deel lezen voor hij aan het tweede stuk begint.

Bij het uitpakken van de ZX Spectrum heeft u het volgende aangetroffen:

1. Dit boek.

2. De computer. Deze heeft drie ingangen voor een jackplug (aangegeven met 9V DC IN, EAR en MIC), een TV-aansluiting en een lange aansluiting voor extra randapparatuur. De computer heeft geen schakelaars. Om hem aan te zetten hoeft u alleen de stekker in het stopcontact te steken.

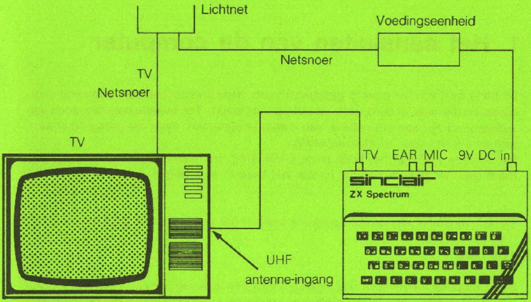
3. Een voedingseenheid. Deze zet de stroom van het lichtnet om in een vorm die de ZX Spectrum gebruikt. Als u uw eigen voedingseenheid wilt gebruiken, moet deze 9 volt wisselspanning bij 1.4 A ongeregeld geven.

4. Een antennekabel van ongeveer twee meter lang, die de computer met de televisie verbindt.

5. Een dubbele kabel van ongeveer 75 cm lang met jack pluggen van 3.5 mm aan de uiteinden. Deze verbinden de computer met een cassetterecorder.

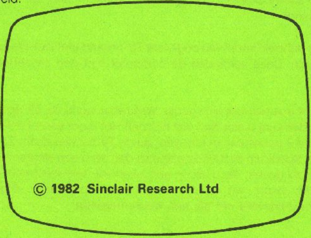
U heeft ook een televisie nodig. Weliswaar werkt de ZX Spectrum ook zonder een TV, maar dan kunt u niet zien wat hij doet! Het moet een UHF-televisie zijn; Als u geen Nederland 2 ermee kan ontvangen, zal de TV het niet doen. Zoals de naam al zegt, geeft de ZX Spectrum een kleurensignaal dat, als u een kleurentelevisie heeft, een gekleurd beeld zal geven. Als u alleen een zwart/wit- televisie heeft, zullen de kleuren verschijnen als zwart, wit en zes verschillende grijstinten. Afgezien hiervan werkt een zwart/wit-televisie net als een kleurentelevisie.

De verschillende onderdelen van het systeem kunnen nu met elkaar verbonden worden op de volgende wijze:



Figuur 1

Als uw televisie twee antenne-ingangen UHF en VHF heeft, moet de UHF-ingang gebruikt worden.
Steek de stekker in het stopcontact en zet uw televisie aan. U moet nu de televisie afstellen. De ZX Spectrum werkt op kanaal 36, en geeft als hij net is aangezet en goed is afgesteld het volgende beeld:



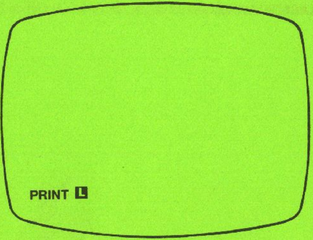
Figuur 2

Als u de computer gebruikt, is het beter om het geluidsvolume van de televisie helemaal uit te zetten.
Als uw televisie een regelbare afstemknop heeft, moet u wellicht deze gebruiken om het beeld van figuur 2 te krijgen. Veel televisies hebben tegenwoordig voorkeuzeschakelaars voor elke zender. Kies een schakelaar die nog niet in gebruik is, en stem deze af op de computer.
Voor gebruik in landen met een ander televisiesysteem dan dat in Engeland en Nederland is een speciaal voor dat systeem ontworpen ZX Spectrum nodig. Engeland gebruikt een UHF-systeem met 625 lijnen en 50 beelden per seconde. Het gebruikt daarnaast een kleurcoderingssysteem dat PAL genoemd wordt. De meeste landen in West-Europa (behalve Frankrijk) gebruiken een soortgelijk systeem, en de computer kan in deze landen gebruikt worden zonder enige aanpassing. Amerika, Canada en Japan gebruiken een totaal verschillend TV-systeem, waardoor een andere versie van de computer nodig is.
Als u de ZX Spectrum uitzet, is alle informatie die erin opgeslagen was, verloren. Een van de manieren om de informatie voor later te bewaren, is om deze op te nemen op een cassettebandje. U kunt ook bandjes kopen die andere mensen habben gemaakt en op deze manier hun programma's gebruiken. De kabel met aan elke kant twee jack-pluggen is bedoeld om een gewone cassetterecorder met de ZX Spectrum te verbinden. Hoofdstuk 8 van deze inleiding vertelt hier meer over.
Nu u de computer aangesloten heeft, wilt u hem natuurlijk gebruiken. De rest van dit eerste deel van het boek vertelt hoe u dat moet doen. Door uw ongeduld heeft u waarschijnlijk al een paar toetsen ingedrukt, en gezien dat daardoor de copyrighttekst verdwenen is. Dit is heel goed: **u kunt de computer niet beschadigen door gewoon toetsen in te drukken.** Ga gerust uw gang. Probeer van alles. Als u vastloopt, kunt u de computer altijd terugkrijgen in de oorspronkelijke staat met de copyrighttekst door de 9V DC IN-plug even uit de computer te halen en er weer in te steken. Dit moet echter wel een laatste uitweg blijven, want op deze manier verliest u alle informatie die in de computer was opgeslagen.

WAARSCHUWING. Gebruik de ZX 16K RAM niet met de ZX Spectrum. Dit werkt niet!

2. Het toetsenbord

Het toetsenbord van de Spectrum lijkt heel erg op dat van een gewone schrijfmachine. De letter- en getaltoetsen zitten op dezelfde plaats, maar elke toets kan meer dan een funktie vervullen. Op een gewone schrijfmachine verschijnen de letters in 'lower case' (kleine letters), of bij gebruik van de SHIFT toets, in 'upper case' (hoofdletters). Bij het Spectrum-toetsenbord is het precies hetzelfde.
Om u te laten zien in welk werkniveau het toetsenbord is, verschijnt een diapositieve (wit op zwart) letter op het scherm. Deze geeft de positie aan van het volgende karakter dat op het scherm zal verschijnen als een toets ingedrukt wordt. De letter knippert om hem te onderscheiden van andere al op het scherm aanwezige karakters. Deze letter wordt een cursor genoemd.
Onmiddellijk na het aanzetten toont de Spectrum een copyright tekst op het scherm. Het indrukken van een willekeurige toets geeft het woord dat onder de letter staat weer op het scherm. Zo'n woord heet een 'keyword' (sleutelwoord). De computer drukt dit sleutelwoord af en niet de letter, omdat op dit moment de computer een opdracht van u verwacht. Met een opdracht vertelt u de computer wat hij moet doen. Elke opdracht *moet* met een keyword beginnen. In tegenstelling tot de meeste andere computers geeft de Spectrum u de mogelijkheid hele keywords op te geven door het indrukken van een enkele toets.
Bijvoorbeeld: Als de **P** toets ingedrukt wordt onmiddellijk na het aanzetten, verschijnt het keyword **PRINT** op het scherm. Het " symbool staat ook op de **P** toets. Om dit te krijgen moet u twee toetsen tegelijkertijd indrukken: hou de **SYMBOL SHIFT** toets in, rechts onderaan het toetsenbord, en druk, terwijl u deze toets ingedrukt houdt, de **P** toets in.



Figuur 3

De cursor verandert nu in een **L** omdat de computer nu een letter verwacht. Tik in:**"Hallo"**. Zet, als er andere tekst op het scherm staat, de computer uit (9V-plug even uittrekken) en begin opnieuw. Gebruik de **CAPS SHIFT** toets om de hoofdletter **H** te krijgen. In het algemeen geldt dat iets dat wit gekleurd boven de toets staat, verkregen wordt met **CAPS SHIFT** en iets dat rood gekleurd staat op de toets verkregen wordt met **SYMBOL SHIFT**. Een opdracht die begint met **PRINT** vertelt de computer dat datgene wat tussen aanhalingstekens staat op het scherm weergegeven moet worden. Om deze opdracht door de computer uit te laten voeren, moet de **ENTER** toets ingedrukt worden. Als dit gedaan is moet het scherm het woord

HELLO

tonen en nog wat andere karakters. (Een knipperend vraagteken duidt op een vergissing ergens. Begin opnieuw als dit gebeurt, en herhaal de oefening). De tekst onderaan het scherm is een mededeling van de computer die weergeeft dat alles goed gegaan is, en dat nu alles "OK" is. Deze mededeling is belangrijk bij het laten lopen van programma's, maar kan op het ogenblik genegeerd worden.
Merk nog iets belangrijks op: De letter O en het getal Ø worden weergegeven door verschillende karakters. Het is belangrijk dat u dit onthoudt. Het getal Ø heeft *altijd* een schuin streepje. De computer ziet de letter O *altijd* als een letter. Druk dus niet de verkeerde toets in. Het getal 1 en de kleine letter l zijn ook verschillend en kunnen niet, zoals bij sommige schrijfmachines, door elkaar gebruikt worden.
Omdat het werkniveau van het toetsenbord zo belangrijk is, is het nuttig om nog even samen te vatten wat er allemaal gebeurt.
De knipperde letter **L** wordt de cursor genoemd. Hij toont waar de computer het eerstvolgende dat u tikt op het scherm zal neerzetten. De cursor is niet altijd een **L**; als u de computer uit en daarna weer aan zet, en u drukt op **ENTER**, verandert de copyrighttekst in een **K** cursor. De letter die gebruikt wordt, vertelt hoe de computer het eerstvolgende dat u intikt zal opvatten. Aan het begin van een regel zal de cursor een knipperde **K** zijn, wat staat voor "keyword". (De copyrighttekst en mededelingen gelden als een knipperde **K**). Een keyword (sleutelwoord) is een van de speciale woorden van de computer. Ze komen voor aan het begin van elke opdracht zodat de computer een algemeen idee heeft over de werking van de opdracht. Omdat de computer een keyword aan het begin van een regel verwacht, wordt bijvoorbeeld een door u ingedrukte **P** toets niet opgevat als een **P**, maar als **PRINT**. U wordt hiervoor gewaarschuwd door een **K** cursor. Als de computer het eerste keyword opgegeven gekregen heeft, verwacht hij niet nog een keyword, dus wordt wat u daarna intikt opgevat als letters. Om dit te laten zien, verandert de computer de cursor in een **L** van "letter".
Deze verschillende manieren waarop de computer het indrukken van een toets opvat, worden modussen of modi genoemd. Een langer woord is "werkniveau". In dit deel zullen we het hebben over keyword (of K-) modus en letter (of L-) modus.
Als u veel hoofdletters wilt intikken zonder de **CAPS SHIFT** te moeten blijven vasthouden, kunt u alle letter als hoofdletters laten verschijnen door eerst **CAPS**

LOCK(**CAPS SHIFT** met **2**). Om te tonen dat dit gebeurd is wordt de **L** cursor vervangen door een knipperende **C** (van "capitals"=hoofdletters). Om kleine letters en de **L** cursor terug te krijgen, drukt u **CAPS LOCK** een tweede keer in.
(Als u **CAPS LOCK** intikt tijdens de keyword modus zult u niet onmiddellijk verschil zien. Het effect wordt echter zichtbaar zodra het sleutelwoord ingetikt is, de computer gaat dan naar C-modus in plaats van L-modus).
Naast de keywords, letters, cijfers en verschillende programmeer- en wetenschappelijke uitdrukkingen, heeft het toetsenbord ook acht grafische karakters. Deze bevinden zich op de cijfertoetsen 1 tot en met 8 en kunnen op dezelfde wijze als letters en cijfers op het scherm gezet worden. Om dit te doen moet het toetsenbord veranderd worden in *grafische* modus. Dit doet u door **CAPS SHIFT** en **9** gelijk in te drukken. Merk op dat de cursor verandert in een **G** . Door de **9** in te drukken, keert u weer terug naar de L-modus.
Er is nog een modus waarin het toetsenbord gebracht kan worden. Dit is de *uitgebreide* modus (extended mode). Deze wordt weergegeven door een **E** cursor. U krijgt deze modus door **CAPS SHIFT** en **SYMBOL SHIFT** gelijk in te drukken. In deze modus kunt u gebruik maken van de meeste wetenschappellijke en programmeerfunkties. Door de twee **SHIFT** toetsen tegelijk in te drukken, keert het toetsenbord weer terug naar de letter, L, modus.
Zelfs als u een zeer ervaren typist(e) of programmeur bent, komen tikfouten voor. Tot nu toe was de enige oplossing hiervoor het uittrekken van de plug. Hoewel dit handig is als er nog maar een opdracht is gegeven, is het bijzonder onhandig als er al een heleboel informatie opgegeven is aan de computer.
Gelukkig kunnen we gebruik maken van de **DELETE** (verwijder) toets om vergissingen te verwijderen. Met bijvoorbeeld de eenvoudige opdracht:

PRINT "HELLO"

kan toch niets mis gaan. Hoewel....

Laten we aannemen dat u vergat om de **SYMBOL SHIFT** toets te gebruiken om de eerste aanhalingstekens te krijgen. Het beeldscherm zou dan het volgende laten zien:

PRINT PHELLO"

De computer zou datgene dat achter **PRINT** komt niet begrijpen, omdat het ontbreken van aanhalingstekens bij de ZX Spectrum aangeven dat een cijfer verwacht moet worden, terwijl een letter ingetikt werd. De verwarring wordt weergegeven door een knipperende **?** aan het einde van de regel.
Gelukkig hoeft u niet alles opnieuw in te tikken. Op de bovenste rij van het toetsenbord zitten vier pijlen die elk een andere kant op wijzen en het woord **DELETE**. Om deze toetsen te kunnen gebruiken, moet u **CAPS SHIFT** indrukken als u een van deze toetsen indrukt. De naar opzij wijzende pijlen verplaatsen de cursor naar links of rechts, en de **DELETE** toets wist het karakter onmiddellijk voor de cursor uit.

Om uw nonsensregel te verbeteren, drukt u ← (**CAPS SHIFT** en **5**) totdat de cursor direkt achter de fout ingetikte **P** staat. Als u de toetsen een of twee seconden vasthoudt, gaan zij automatisch werken terwijl ze een klikkend geluid voortbrengen. Elke toets gaat zichzelf automatisch herhalen als u hem langer dan drie seconden ingedrukt houdt. Tik **DELETE** (**CAPS SHIFT** en **0**) om de verkeerde **P** te verwijderen en tik dan " (**SYMBOL SHIFT** en **P**) om het ontbrekende toe te voegen. Merk op dat er toegevoegd wordt zonder dat iets anders gewist wordt. Probeer de cursor rechts-toets ook, enkel om even te oefenen. Als u echte tikfouten gemaakt heeft, kunt u ze op dezelfde wijze verbeteren. Onthoudt dat u vergissingen niet weg kunt schrijven: U moet ze eerst verwijderen en daarna de correcties aanbrengen.
Als u nu **ENTER** indrukt, schrijft de computer uw bericht bovenaan het scherm, of vlak onder de vorige mededeling als die er nog steeds is.
Een volledige beschrijving van het toetsenbord vindt u in hoofdstuk 1 van deel twee van dit boek: De BASIC- programmeerhandleiding.

3. Cijfers, letters en de computer als rekenmachine

We hebben al gezien hoe we de computer vertellen om letters en grafiek op het beeldscherm te zetten door middel van **PRINT**. We hebben ook gezien dat **ENTER** gebruikt moet worden om de computer te laten weten dat een net ingetikte opdracht uitgevoerd moet worden. Vanaf nu zal in deze handleiding niet meer gesproken worden over het gebruik van **ENTER**. We gaan ervan uit dat u dit gebruikt na elke regel.
Getallen kunnen door de computer nog eenvoudiger dan letters verwerkt worden. In het vorige hoofdstuk zeiden we dat al impliciet door uit te leggen dat de computer een getal verwacht na een **PRINT** als er geen aanhalingstekens worden gebruikt.

Dus als we intikken

PRINT 2

zal het getal **2** op het scherm verschijnen.

Het is mogelijk om letters en getallen door elkaar te gebruiken:

PRINT 2,"ABC"

Merk op dat op het beeldscherm een ruimte tussen **2** en **ABC** staat.

Tik nu

PRINT 2;"ABC"

en dan

PRINT 2 "ABC"

Een komma tussen de verschillende onderdelen na een **PRINT** zorgt voor een scheiding van *16* kolommen. Het gebruik van een puntkomma geeft geen scheiding en het weglaten van een leesteken geeft een vergissing.

PRINT kan ook gebruikt worden met de wiskundige funkties op het toetsenbord. De ZX Spectrum kan als elektrische rekenmachine gebruikt worden.

Bijvoorbeeld:

PRINT 2+2

Het antwoord verschijnt bovenaan het scherm. Vergelijk dit met:

PRINT "2+2"

Het is mogelijk deze twee effecten te combineren tot iets nuttigers. Probeer

PRINT "2+2=";2+2

Probeer ook wat andere rekenkunde:

PRINT 3-2
PRINT 4/5
PRINT 12*2

De * wordt gebruikt als een vermenigvuldigingsteken in plaats van een X om verwarring met de letter **x** te voorkomen. Het teken / wordt gebruikt voor het deelteken ÷ .

Experimenteert u vooral met een stel verschillende berekeningen. Probeert u ook negatieve getallen of getallen met cijfers achter een komma.

Als u genoeg oefent, zodat u alle 22 lijnen van het scherm gebruikt, zult u iets interessants zien gebeuren; alles zal een regel opschuiven en de bovenste regel valt weg. Dit wordt het *rollen* (*scrolling*) van het beeld genoemd.

Berekeningen worden niet altijd in de door u verwachte volgorde uitgevoerd. Probeer bijvoorbeeld

PRINT 2+3*5

U zou kunnen denken dat dit zorgt voor het nemen van 2, 3 daarbij opgeteld, samen 5, en dit vermenigvuldigd met 5: totaal 25. Maar dat is niet zo. Vermenigvuldigingen - en delingen - worden uitgevoerd vóór optellingen en aftrekkingen. De uitdrukking '2+3*5' betekent 'neem 3 en vermenigvuldig dit met 5, produkt 15; en tel dat op bij 2, totaal 17'. Het antwoord op het scherm moet dus 17 zijn.

Omdat vermenigvuldigingen en delingen het eerst gedaan worden, zeggen we dat ze voorrang hebben boven optellen en aftrekken. Onderling hebben vermenigvuldigingen en delingen dezelfde rangorde zodat ze van links naar rechts worden uitgevoerd. Als ze gedaan zijn, blijven de optellingen en aftrekkingen over. Deze hebben onderling ook dezelfde rangorde en worden dus ook van links naar rechts uitgevoerd.

Laten we eens kijken hoe de computer dit uitwerkt:

PRINT 20-2*9+4/2*3

a. 20-2*9+4/2*3
b. 20-18+4/2*3 } Eerst de vermenigvuldigingen en delingen van links naar rechts
c. 20-18+2*3
d. 20-18+6
e. 2+6 } en dan de optellingen en aftrekkingen
f. 8

Hoewel u alleen hoeft te weten of een bepaalde bewerking een hogere of een lagere rangorde heeft dan de andere, doet de computer dit door elke bewerking een rangordegetal tussen de 1 en 16 te geven: * en / hebben rangorde 8, en + en - hebben rangorde 6.
Deze volgorde van het uitvoeren van bewerkingen is absoluut. U kunt de volgorde echter beïnvloeden door haakjes te gebruiken; alles tussen haakjes wordt eerst uitgevoerd, en dan verder behandeld als een enkel getal. Dus

PRINT 3*2+2

geeft als antwoord 6+2=8, maar

PRINT 3*(2+2)

geeft als antwoord 3*4=12

Het is soms handig om de computer dit soort sommen op te geven want telkens als de computer een getal van u verwacht kunt u hem een rekenkundige bewerking opgeven en hij werkt het resultaat zelf uit. Er zijn zo weinig uitzonderingen op deze regel dat die telkens expliciet vermeld zullen worden.
U kunt getallen ook met decimale punten opschrijven (Let op! Gebruik niet de komma, maar de *punt*), of in wetenschappelijke notatie - tamelijk gebruikelijk bij zakrekenmachines. U schrijft in deze methode na een gewoon getal (met of zonder decimale punt) een exponent die bestaat uit de letter **e**, dan misschien een -, en dan een getal. De exponent schuift de decimale punt naar rechts (of naar links bij een negatieve exponent), waardoor het originele getal (voor de exponent) een aantal malen wordt vermenigvuldigd (of gedeeld) met (of door) 10. Bijvoorbeeld:

2.34e0=2.34
2.34e3=2340
2.34e-2=0.0234 etc.

(Probeer voorgaande voorbeelden op de computer uit). Dit is een van de gevallen waarbij u een getal niet kunt vervangen door een bewerking: U kunt bijvoorbeeld niet schrijven

(1.34+1)e(6/2).

U kunt ook uitdrukkingen hebben waarvan de waarden geen getallen maar strings (slierten) van letters zijn. De eenvoudigste vorm daarvan heeft u al verschillende keren gezien; de string letters die opgeschreven werd met aanhalingstekens er omheen. Dit was helemaal analoog aan het opschrijven van een enkel getal. Wat u nog niet gezien heeft, is het gebruik van + met strings (-,* en / kunnen niet gebruikt worden bij strings, dus het probleem van de rangorde speelt hierbij niet). Door strings op te tellen worden ze achter elkaar gezet; probeer

PRINT "fri"+"es stamboekvee"

U kunt zoveel strings in een bewerking optellen als u wilt. Indien u dat wilt, kunt u zelfs haakjes gebruiken.

4. Een paar eenvoudige opdrachten

In het geheugen van de computer kan van alles opgeslagen worden. Tot nu toe hebben we gezien dat de **PRINT** opdracht de mogelijkheid biedt om letters, getallen en de resultaten van berekeningen met zowel letters als getallen op het scherm te zetten.
Als we de computer willen vertellen om een bepaald getal, of een string letters te onthouden, moeten we wat geheugenruimte reserveren voor dat doel.
De meeste zakrekenmachines hebben hiervoor een aparte toets 'memory' die gebruikt wordt om getallen voor later te onthouden (memory=geheugen). Uw computer kan dit veel beter. Hij kan zoveel van dit soort 'opbergdoosjes' hebben als u wilt, en u kunt elk doosje een aparte naam geven.
Laten we als voorbeeld aannemen dat u uw leeftijd wilt onthouden! Hiervoor wordt de **LET** opdracht gebruikt (**LET** is het keyword op de **L** toets). Stel u bent 34 jaar

LET leeftijd=34

Wanneer de opdracht **LET** gebruikt wordt, wordt een geheugenplaatsje de naam 'leeftijd' gegeven en in die geheugenplaats wordt de waarde 34 opgeslagen. Om deze informatie uit deze geheugenplaats te lezen, tikt u

PRINT leeftijd

en het getal 34 komt te voorschijn. Het is heel eenvoudig om de inhoud van de geheugenplaats met de naam 'leeftijd' weer te veranderen. Tik:

LET leeftijd=56

en tik dan:

PRINT leeftijd

Dit keer verschijnt 56 op uw scherm. 'leeftijd' wordt een variabele genoemd, omdat de waarde ervan kan variëren, variabel is. Het is mogelijk om het weergeven van een bericht direkt op het scherm en het weergeven van de waarde van een variabele te combineren. Tik:

PRINT "Uw leeftijd is ";leeftijd

De computer kan voor nog veel meer zaken gebruikt worden dan alleen het onthouden van getallen met een naam. Slierten letters (strings) kunnen ook onthouden worden. Om te kunnen onderscheiden tussen een getalvariabele en een zogenaamde stringvariabele wordt het dollarteken **-$-** gebruikt aan het eind van de stringvariabele naam. Als u bijvoorbeeld de string letters

"Uw leeftijd is "

wilt bewaren kunt u deze noemen:

a$

(de namen van stringvariabelen mogen uit maar een letter, voor het dollarteken, bestaan). Tik nu:

LET a$="Uw leeftijd is "

Als u nu tikt:

PRINT a$

komt de string terug op het beeldscherm.
Als de computer sinds het begin van dit hoofdstuk niet uit geweest is, kunt u

PRINT a$;leeftijd

intikken en kijken wat er gebeurt.
Er zijn ook andere manieren om informatie in het geheugen van de computer te stoppen zonder de **LET** opdracht te gebruiken.
De **INPUT** opdracht bijvoorbeeld vertelt in zijn eenvoudigste vorm dat de computer informatie van het toetsenbord kan verwachten. In plaats van elke keer **LET** enz. te tikken kunt u tikken

INPUT leeftijd

Zodra de **ENTER** toets ingedrukt is, zal een knipperende **L** cursor op het scherm verschijnen. Dit betekent dat de computer op informatie van u wacht. Tik uw leeftijd in en druk op de **ENTER** toets. Hoewel het lijkt alsof er niets gebeurd is, heeft de variabele nu de waarde gekregen die u ingetikt heeft. Als u

PRINT leeftijd

tikt, zult u zien dat dit klopt.

Laten we deze opdrachten eens allemaal combineren. Tik:

LET b$="Hoe oud bent u?"
LET a$="Uw leeftijd is "
INPUT (b$);leeftijd:PRINT a$;leeftijd

Merk op dat de laatste regel uit twee opdrachten bestaat die gescheiden zijn door een dubbele punt.

INPUT (b$);leeftijd

betekent hetzelfde als

INPUT "Hoe oud bent u?";leeftijd

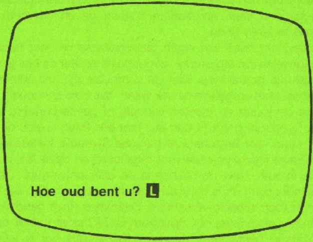
5. Eenvoudig programmeren

Tot nu toe hebben we de computer verteld wat er moest gebeuren door direkt iets op het toetsenbord in te tikken. Hoewel het mogelijk is verschillende opdrachten te combineren, is met deze methode het aantal mogelijkheden beperkt.
Het geweldige aan computers is dat ze programmeerbaar zijn. Dit betekent dat we een computer een serie opdrachten kunnen geven om hem verschillende dingen in volgorde te laten doen.
Elke computer heeft zijn eigen taal waarmee we met hem kunnen communiceren. Sommige talen zijn erg simpel - zodat de computer de taal gemakkelijk begrijpt. Helaas zijn talen die gemakkelijk voor de computer zijn, moeilijk voor mensen. In sommige opzichten is het omgekeerde ook waar - talen die eenvoudig voor ons zijn, zijn moeilijk voor de computer en moeten vertaald of geïnterpreteerd worden.
De ZX Spectrum gebruikt een taal met een hoog niveau die BASIC genoemd wordt. BASIC staat voor Beginners All-purpose Symbolic Instruction Code (multi-functionele symbolische instruktie code voor beginners) en deze taal is in 1964 op de universiteit van Dartmouth, New Hampshire in de USA ontworpen. De taal wordt op bijna alle persoonlijke computers gebruikt, maar hoewel de taal op alle machines ongeveer gelijk is, zijn er toch subtiele verschillen. Daarom is deze handleiding speciaal voor de ZX Spectrum geschreven. ZX Spectrum BASIC staat echter niet ver af van het (niet in werkelijkheid bestaande) standaard-BASIC dus het is niet zo moeilijk om andere BASIC-programma's aan te passen, zodat ze op de ZX Spectrum werken. In tegenstelling tot andere BASICs staat ZX Spectrum BASIC niet toe dat de opdracht **LET** wordt weggelaten bij het toekennen van een waarde aan een variabele.
Het aantal opdrachten dat in de computer opgeslagen kan worden, is niet onbeperkt. Zodra de limiet bereikt is, geeft de ZX Spectrum dit aan met een zoemer.
Als u in BASIC programmeert is het belangrijk dat u de computer te kennen geeft in welke volgorde de opdrachten uitgevoerd moeten worden. Elke regel met opdrachten heeft dan ook een nummer aan het begin. Het is gebruikelijk om te beginnen bij nummer 10 en het nummer telkens te verhogen met 10 bij elke nieuwe regel. Op deze wijze kunnen later eventueel nog regels tussen de al bestaande regels gevoegd worden.
Laten we eens een eenvoudig programma bekijken. Bekijk de opdrachten aan het einde van het vorige hoofdstuk nog eens. Als we deze opdrachten nog een keer aan de computer willen geven moeten ze allemaal opnieuw getikt worden. Een *programma* maakt dit overbodig.
Tik het volgende in. Vergeet de **ENTER** niet na elke regel.

```
10 LET b$="Hoe oud bent u? "
20 LET a$="Uw leeftijd is "
30 INPUT (b$);leeftijd
40 PRINT a$;leeftijd
```

Merk op dat u nergens spaties hoeft in te tikken, behalve binnen de aanhalingstekens. Er gebeurt nu helemaal niets tot we de computer opdracht geven om met het programma te gaan werken. Dat gebeurt door middel van **RUN** (het sleutelwoord op de **R**).

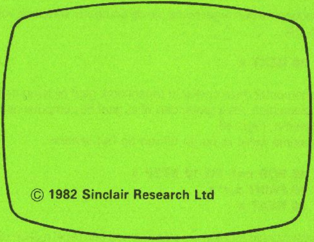
Tik deze opdracht en kijk goed wat er gebeurt.



figuur 4

Het kan u opgevallen zijn dat er, telkens nadat een regel is ingevoerd, een naar rechts wijzend pijltje op het scherm staat. Dit pijltje geeft de laatste ingevoerde regel aan. Als u het programma nog een keer wilt zien, drukt u op **ENTER** (of u gebruikt **LIST**). U kunt **RUN** zo vaak gebruiken als u wilt om het programma te laten uitvoeren. Als u het programma niet meer nodig heeft, kunt u het verwijderen door de **NEW** opdracht te gebruiken. Deze opdracht wist het programma uit het geheugen en geeft u een leeg geheugen om opnieuw te beginnen.

Tik **NEW** en dan **LIST** en kijk wat er gebeurt.



figuur 5

Samenvattend:
Als u een opdracht vooraf laat gaan door een getal, zegt dit de computer dat het niet om zomaar een getal gaat, maar om een programmaregel. De computer voert dit daarom niet uit, maar slaat de regel op voor later gebruik.
De ZX Spectrum is zo behulpzaam om alle regels die u ingetikt heeft op het scherm te zetten (dit heet listen). Er staat een > voor de laatste door uw ingetikte regel.
De computer voert deze regels niet gelijk uit, maar slaat ze intern op voor later gebruik.
Om de computer deze regels te laten uitvoeren moet u de opdracht **RUN** geven.
Als u alleen **ENTER** gebruikt, krijgt u de listing weer terug.
Laten we nog eens een eenvoudig programma bekijken. Dit programma is ietsje wiskundiger; het print alle kwadraten van de getallen van 1 tot en met 10 (een kwadraat van een getal is dat getal vermenigvuldigd met zichzelf).
Om de getallen van 1 tot en met 10 te krijgen, introduceren we een nieuw concept in BASIC programmeren. Dat is de methode waarmee we de computer kunnen laten tellen. Eerder hebben we gezien dat we getallen in het geheugen kunnen opslaan door er een 'naam' aan te verbinden, ook wel het toekennen van een waarde aan een variabele genoemd. De variabele **x** moet beginnen met 1, telkens 1 hoger worden en eindigen bij 10. Dit effect wordt bereikt door de opdracht **FOR .. TO .. STEP**.
Om dit programma in te voeren, tikt u **NEW** om het vorige programma te verwijderen en het volgende

```
10 FOR x=1 TO 10 STEP 1
```

(Normaal gesproken kan **STEP 1** weggelaten worden als het tellen met 1 tegelijk gaat).
De volgende regel moet de computer vertellen wat er met **x** moet gebeuren, ongeacht de waarde daarvan. Tik dus:

```
20 PRINT x,x*x
```

Tot slot hebben we een regel nodig die de computer vertelt de volgende waarde voor **x** te nemen. Tik

```
30 NEXT x
```

Zodra de computer deze opdracht tegenkomt, gaat hij terug naar regel 10 en herhaalt de serie opdrachten. Als **x** groter dan 10 is, gaat de computer naar de volgende regel in het programma, regel 40.
Het programma moet er nu zo uitzien op het scherm:

```
10 FOR x=1 TO 10 STEP 1
20 PRINT x,x*x
30 NEXT x
```

Voor de volledigheid moeten we eigenlijk nog een regel maken die de computer vertelt dat het programma afgelopen is als **x**=10:

```
40 STOP
```

Als het programma nu uitgevoerd wordt (ge**RUN**d wordt - van 'run' = hollen/lopen), moeten er twee kolommen verschijnen. De eerste met waarden van **x**, en de tweede met waarden van **x*x** of **x** kwadraat. Het is mogelijk deze kolommen van een 'kop' te voorzien door een regel als

```
5 PRINT "x","x*x"
```

Merk op dat deze regel weliswaar ingevoerd werd *na* alle andere regels, maar toch door middel van het regelnummer door de computer op de juiste plaats gezet werd. Probeert u zelf programma's te schrijven met andere wiskundige funkties. Als u twijfelt over het gebruik van een bepaalde funktie, sla deze dan op in deel twee van dit boek; de BASIC- programmeerhandleiding.

6. Het gebruik van de cassetterecorder

Het is nogal vermoeiend om telkens als u de computer wilt gebruiken de programma's helemaal in te moeten tikken. De ZX Spectrum heeft de faciliteiten om programma's op te nemen op magnetische banden op een normale huis-, tuin- en keukencassetterecorder. Als u een programma in het geheugen heeft zitten, kunt u het bewaren op de hier beschreven methode.
Als u een programma op cassette heeft opgenomen, kunt u het later weer in het geheugen laden.
De meeste cassetterecorders zijn goed. Voor zover het de computer betreft zijn de goedkope eenvoudige mono-cassetterecorders minstens zo goed als de dure stereotypen, en geven ze minder storingen. Een bandteller is een nuttig snufje.
De cassetterecorder moet een ingang hebben voor het gebruik van een microfoon, en een uitgang voor een koptelefoon (als deze er niet is, kunt u de externe luidsprekeruitgang proberen). De in- en uitgangen moeten geschikt zijn voor 3.5 mm jack-pluggen, zodat de pluggen aan het meegeleverde snoertje passen. Andere soorten stekkers geven vaak een te zwak signaal voor de computer.
Elke cassette is goed, maar low-noise bandjes werken vaak beter. Heeft u de geschikte cassetterecorder, sluit hem dan met de computer aan door middel van het snoer dat met de ZX Spectrum meegeleverd wordt. Een draad moet de microfooningang verbinden met de 'MIC'-uitgang aan de achterkant van de computer, de andere draad moet de koptelefoonuitgang van de cassetterecorder verbinden met de 'EAR'-ingang van de computer. (U kunt de computer niet beschadigen door de draden verkeerd om aan te sluiten.)
Wanneer u de **SAVE** opdracht gebruikt om een programma op band op te nemen, moet u zich ervan vergewissen dat een van de twee stekkers van de draad die de 'EAR'-uitgangen verbindt, verwijderd is. Als u dit vergeet, krijgt u alleen een monotone brom. De reden hiervan is dat als de cassetterecorder aan het opnemen is, het inkomende signaal versterkt via de koptelefoonuitgang weer naar buiten gestuurd wordt. Als dit signaal dan weer in de computer komt, vormt het een lus die gaat oscilleren en daardoor het oorspronkelijke signaal onderdrukt.
Maak een programma voor de computer, bijvoorbeeld het vierkantenprogramma uit het vorige hoofdstuk, en tik dan;

SAVE "vierkant"

vierkant is de naam waaronder u het programma op de cassette bewaart. U mag maximaal 10 karakters voor een naam gebruiken. Deze karakters moeten ofwel letters

ofwel cijfers zijn.

De computer geeft hierna de melding **Start tape then press any key** (start de cassette en druk op een willekeurige toets). We gaan eerst een keertje 'droog' oefenen, zodat u kunt zien wat er allemaal gaat gebeuren: Zet de cassetterecorder niet aan, maar druk wel op een willekeurige toets, en let op de rand van het televisiescherm. U zult allerlei patronen van gekleurde horizontale strepen zien.

5 seconden lang rode en lichtblauwe strepen, ongeveer een centimeter breed en langzaam omhoog lopend

Een korte uitspatting van blauwe en gele strepen.

1 seconde lang is alles weer normaal.

2 seconden lang weer het rood/blauwe patroon

en tenslotte 1 seconde het blauw/gele patroon.

Probeer dit nog een paar keer uit, totdat u het hiervoorstaande herkent. De informatie wordt in twee blokken weggeschreven, en beide blokken hebben een introduktie die correspondeert met het rode en blauwe patroon, en de informatie zelf die overeenkomt met het blauw/gele patroon. Het eerste blok bevat de naam en wat diverse informatie over het op te nemen programma. Het tweede blok bevat het programma en de variabelen. Het witte deel ertussen is slechts een korte pauze.

Laten we het signaal eens daadwerkelijk op een cassette zetten.

1. Zet de cassette zo dat er een blanko stuk klaarstaat of een stuk waarvan u niet erg vindt dat het gewist wordt.

2. Tik

   SAVE "vierkanten" (en **ENTER**)

3. Start de cassetterecorder, zodat deze gaat opnemen.

4. Druk op een willekeurige toets van de ZX Spectrum.

5. Kijk weer naar het beeldscherm. Als de computer klaar is (met de melding **0 OK, 0:1**) stopt de cassetterecorder.

Om er zeker van te zijn dat alles gewerkt heeft, kunt u het signaal op de cassette vergelijken met de in de computer aanwezige informatie door de **VERIFY** (controleer-) opdracht.

1. Zet de volumeregelaar van de cassetterecorder ongeveer halverwege en sluit de 'EAR'-draad weer aan.

2. Spoel de cassette terug tot een punt iets voor het punt waar u net begonnen bent met opnemen.

3. Tik

VERIFY "vierkanten"

(**VERIFY** is een **SHIFT R** in de extended modus).

4. Laat de cassetterecorder afspelen.

De televisierand zal afwisselend rood en lichtblauw zijn totdat de band bij de opname gekomen is. Dan ziet u hetzelfde patroon als bij het opslaan van het programma. In de tweede pauze in het midden wordt **Program vierkanten** op het scherm geschreven. Als de computer iets op band zoekt, wordt de naam van alles wat hij tegenkomt op het scherm afgebeeld. Als u elk patroon gezien heeft en de computer stopt met de melding **0 OK, 0:1** is uw programma veilig op band weggeschreven. U kunt dan het volgende deel overslaan. Als er iets mis gegaan is, kunt u de volgende vragen doorlopen.

Is uw programma veilig opgenomen?

Is de naam verschenen?
Niet? Dan is het programma niet goed opgenomen, of wordt het programma niet goed ingelezen. U moet eerst uitzoeken welk van deze twee mogelijkheden van toepassing is. Om uit te zoeken of het programma goed opgenomen is, spoelt u de band terug tot iets voor de plaats waar u begon met opnemen. Speel de band nu af en luister naar het signaal via de luidspreker van de cassetterecorder. (waarschijnlijk moet u de stekker uit de koptelefoonuitgang verwijderen). De rood/blauwe inleiding geeft een heldere stabiele hoge toon, en het blauw/gele informatiedeel geeft een onaangenaam geluid: een beetje als een morsesignaal in een wervelstorm. Allebei de signalen moeten tamelijk hard klinken. Als u de volumeknop op de hoogste stand zet, moet het signaal normale conversatie onmogelijk maken. Als u deze signalen niet hoort, is het programma helemaal niet opgenomen. Controleer of u de juiste stekker in de juiste in- /uitgang had gestoken. De 'MIC'-kanalen moeten met elkaar verbonden zijn, de 'EAR'-kanalen juist niet! Sommige cassetterecorders hebben de eigenaardigheid dat de stekker niet goed contact maakt als u hem er helemaal insteekt. Probeer de stekker een heel klein stukje terug te trekken. U voelt meestal wel wanneer de stekker precies 'goed' valt. Controleer gelijk of u niet geprobeerd heeft op te nemen op het stukje plastic band aan het begin van de cassette. Heeft u dit alles geprobeerd, save het programma dan nog een keer.
Als u het geluid wel goed hoorde, is het programma waarschijnlijk wel goed opgenomen, en moet u de fout in het teruglezen zoeken.

Controleer de draden nog een keer, en controleer het afspeelniveau. Als dit te laag is, 'hoort' de computer het signaal niet goed, en ziet u niet de juiste patronen op het beeldscherm. Als het signaal te luid is, wordt het signaal vervormd. In dat geval kunt u het signaal door de luidspreker van de computer horen. Tussen deze twee uitersten in liggen nog vele mogelijkheden. Probeert u door experimenteren de juiste waarden te vinden.
Het volgende probleem kan zijn dat de computer het programma vindt, de naam op het scherm zet, maar toch het programma niet inleest. Enkele mogelijkheden zijn:
U heeft de naam verkeerd ingetikt, ofwel bij **SAVE** (de computer geeft de verkeerd getikte naam op het scherm) ofwel bij **VERIFY**: de computer zal het programma negeren en doorgaan met rood/blauw knipperen.
Er kan ook nog een echte fout op de band staan. De computer meldt dan **R tape loading error** (fout bij het inlezen van de band). De computer geeft hiermee aan dat het programma verkeerd op de band staat. **SAVE** het programma nog een keer.
Het is mogelijk dat de volumeafregeling van de cassetterecorder niet goed is; Het kan er echter niet ver naast zijn, want de computer kon het eerste blok wel goed lezen.

Laten we nu aannemen dat u het programma opgenomen heeft en dat de controle goed verliep.
Het weer inlezen gaat op dezelfde manier als het controleren, maar dan met de opdracht

LOAD "vierkanten"

in plaats van

VERIFY "vierkanten"

LOAD vindt u op de **J** toets. Omdat het programma goed gecontroleerd is door **VERIFY** zult u geen problemen ondervinden met het inlezen.
LOAD verwijdert het oude programma (en variabelen) in de computer voordat het nieuwe programma ingelezen wordt.
Nadat een programma ingelezen is, zorgt **RUN** voor het uitvoeren van het programma.
Het is mogelijk voorbespeelde cassettes met programma's te kopen. Deze moeten wel speciaal voor de ZX Spectrum geschreven zijn: Verschillende soorten computers hebben verschillende methoden om programma's op te nemen, dus uitwisselen van bandjes is niet mogelijk.

Als uw cassette meer dan een programma op dezelfde kant heeft staan, zullen ze allemaal een naam hebben. U kunt kiezen uit de verschillende programma's door de **LOAD** opdracht. Als u bijvoorbeeld alleen het programma 'helicopter' wilt, tikt u

LOAD "helicopter"

(**LOAD""** betekent: **LOAD** het eerste programma dat je tegenkomt. Dit kan erg handig zijn als u zich de naam van een programma niet meer kunt herinneren.)

7. Kleuren

Een van de voornaamste redenen om de ZX Spectrum te kopen was ongetwijfeld de mogelijkheid om kleuren te gebruiken. Het televisiescherm is verdeeld in twee gebieden. Het buitenste deel wordt de **BORDER** (rand) genoemd. Het binnenste deel is **PAPER** (papier). Het is mogelijk de kleuren van deze twee gebieden naar wens te veranderen, zowel door een direkte opdracht als door een programmaregel.
De ZX Spectrum heeft acht kleuren ter beschikking, welke Ø tot en met 7 genummerd zijn. Hoewel het lijkt of de kleuren door elkaar staan, staan ze op volgorde van donkerte (wit, via verschillende grijstinten, tot zwart).
Hier een lijst van alle kleuren met de getallen die u moet gebruiken om naar een kleur te verwijzen

Ø	zwart
1	blauw
2	rood
3	paars
4	groen
5	lichtblauw
6	geel
7	wit

Als de computer aangezet wordt, werkt het systeem in zwart en wit. De standaardwaarde voor **BORDER** en **PAPER** is dus 7, wit. De kleur van elk karakter op het scherm wordt bepaald door de **INK** opdracht. Deze staat standaard op Ø, zwart. De drie waarden die de kleuren op het beeldscherm bepalen worden altijd standaard door de computer ingesteld.
Maar u kunt deze waarden wel zelf veranderen. Tik bijvoorbeeld

BORDER 2

Als u niet vergeet om de **ENTER** toets in te drukken, verandert de rand van het scherm van wit in rood. De rand houdt tevens het onderste deel van het scherm waar de opdrachten ingetikt worden in. Tik nog wat getallen om te zien hoe de kleuren veranderen.
Probeer nu het binnenste deel van het scherm te veranderen door

PAPER 5

De **PAPER** opdracht is een van de opdrachten die u in de extended mode (uitgebreide modus) kunt krijgen. U tikt zowel **CAPS SHIFT** als **SYMBOL SHIFT** tegelijk in. U vindt **PAPER** dan door een C met shift. Als u nu de **ENTER** toets twee keer indrukt, verandert het scherm in lichtblauw. De eerste **ENTER** verwijdert de oude **PAPER** opdracht die al in de computer zat. Pas als de tweede **ENTER** ingedrukt wordt (waardoor de computer een programma **LIST**, en daarbij de informatie voor het beeldscherm herformeert) wordt de nieuwe **PAPER** kleur gebruikt. Als u een kleurentelevisie gebruikt, en de kleur is niet veranderd, dan klopt de afstelling van de kleurknoppen op de televisie niet. Probeer deze beter af te stellen, of probeer anders de fijnafstemming te verbeteren.

De **INK** opdracht is vergelijkbaar met de **PAPER** opdracht, en bestuurt de kleuren van de karakters die op het **PAPER** deel van het scherm komen te staan. Natuurlijk verschijnt er niets op het beeldscherm als **PAPER** en **INK** dezelfde kleur hebben! De **BORDER, PAPER** en **INK** opdrachten kunnen in programma's gebruikt worden. Het volgende programma laat de verschillende kleurmogelijkheden zien.

```
10 FOR x=0 TO 7
20 BORDER x
30 PAPER 7-x: CLS
40 PAUSE 50
50 NEXT x
```

Dit programma gaat, na een **RUN**, langs de acht kleuren, waarbij de **PAPER** en **BORDER** telkens afsteken. De **CLS** opdracht na **PAPER** dwingt de computer om het beeldscherm opnieuw op te bouwen, waardoor de nieuwe **PAPER** kleur gebruikt kan worden. De **PAUSE** (pauze) opdracht stopt het programma gedurende 1 seconde, zodat we kunnen zien wat er gebeurt (probeer het programma eens zonder deze opdracht). Om de **INK** opdracht te demonstreren, kunt u het volgende programma intikken (na een **NEW**).

```
10 BORDER 7
20 PAPER 1
30 INK 4
40 PRINT "Groene karakters op blauwe achtergrond"
```

Er zijn nog meer opdrachten die gebruikt kunnen worden voor het manipuleren van de kleuren van de ZX Spectrum. Deze worden in het tweede deel van dit boek behandeld.

8. Geluid

De ZX Spectrum kan een oneindige variatie aan geluiden maken. De frequentie van de toon en de tijdsduur kunnen door de gebruiker bestuurd worden. De opdracht **BEEP** wordt gebruikt om de computer te vertellen een geluid voort te brengen. **BEEP** is een opdracht in de uitgebreide modus, en staat op de **Z** toets.
De 'centrale' frequentie van de **BEEP** opdracht is de midden-C. De frequentie kan beïnvloed worden en elke noot kan verkregen worden door deze uit te drukken in halve noten of delen van halve noten onder of boven deze centrale frequentie. Als u tikt

```
BEEP 2,Ø
```

laat de computer een geluid in midden-C gedurende twee seconden horen.
De twee getallen bepalen hoe het geluid klinkt. Het eerste getal geeft de lengte van de toon in seconden, de tweede geeft de toonhoogte in halve noten boven de midden-C. Het nummer voor de midden-C is Ø, voor C#is het 1, voor D is het 2, en zo verder tot de volgende C die de waarde 12 heeft. (twaalf halve tonen vormen een octaaf). U kunt nog hogere waarden gebruiken: Hoe hoger de waarde, hoe hoger de toon.

Probeer

```
BEEP 1,4: BEEP 1,2: BEEP 2,Ø
```

U zou nu de eerste drie noten van 'drie blinde muizen' moeten horen. Omdat u zeer veel **BEEP**s op deze manier achter elkaar kunt zetten, zou u, als u geduld heeft, op deze wijze een heel liedje kunnen componeren.
(Dubbele punten verbinden niet alleen **BEEP**s; U kunt ze gebruiken voor alle opdrachten.)
Om een wat ingewikkelder opdracht te maken, kunt u een zingende kameleon maken door **BEEP** en **BORDER** te mengen:

```
BORDER 1: BEEP 1,14: BORDER 3: BEEP 1,16: BORDER 4: BEEP 1,12:
BORDER 6: BEEP 1,Ø: BORDER 5: BEEP 4,7: BORDER 1
```

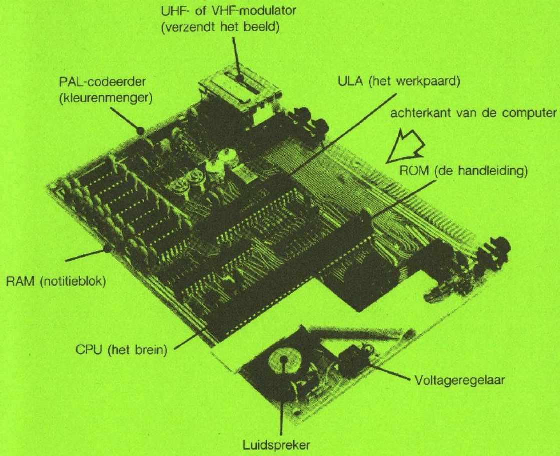
(U hoeft zich geen zorgen te maken over het feit dat deze opdrachten meer dan een regel in beslag nemen: de computer trekt zich hier niets van aan.)
Een kort programma om een hele serie noten te spelen, kan er als volgt uit zien:

```
10 FOR x=0 TO 24
20 BEEP 2,x
30 NEXT x
```

Er zijn nog veel meer mogelijkheden met deze opdracht. Zie ook hiervoor deel twee van dit boek.
Noten lager dan de midden-C worden aangegeven met negatieve waarden.

9. De binnenkant

De foto hieronder toont u de binnenkant van een ZX Spectrum.



Zoals u ziet zijn alle onderdelen aangegeven door een drie letter afkorting. De zwarte rechthoekige stukken plastiek met heel veel metalen pootjes zijn de geïntegreerde circuits (IC's - de chips) die al het werk doen. In elke IC zit een klein vierkant stukje silicone dat door middel van draden met de metalen poten is verbonden. Op die siliconechip bevinden zich duizenden transistoren die samen de elektronische circuits vormen waaruit de computer bestaat. (Een computer is niets anders dan een verzameling elektronische circuits.)
Het brein van de computer wordt gevormd door de processorchip, meestal de CPU (Central Processor Unit - centrale verwerkingseenheid). Deze CPU is een Z80A, een snellere versie van de populaire Z80.
De processor bestuurt de computer, doet het rekenwerk, kijkt welke toetsen ingedrukt zijn, bepaalt wat er gaat gebeuren, en besluit in het algemeen over de volgende stap van de computer. Hoewel deze processor dus heel knap is, kan hij dit alles niet alleen. De processor weet niets van BASIC of van rekenkunde met een decimale punt. Alle instructies hierover moet hij weer krijgen van een andere chip, de ROM (Read Only Memory - enkel lees- geheugen). De ROM bestaat uit een lange lijst van opdrachten die samen een computerprogramma vormen. Dit programma vertelt de processor wat er in alle denkbare gevallen moet gebeuren. Het programma is niet in BASIC maar in zogenaamd Z80 machinecode geschreven, en bestaat uit een lange serie getallen. Er zijn in totaal 16384 (16*1024) van deze getallen. Daarom wordt de BASIC van de ZX Spectrum een 16K BASIC genoemd - 1K is 1024.
Hoewel er vergelijkbare chips in andere computers zitten, is deze volgorde van instrukties uniek voor de ZX Spectrum en speciaal voor deze machine geschreven. De acht chips hiernaast zijn bedoeld voor het geheugen. Dit is het RAM (Random Access Memory - vrij toegankelijk geheugen) waar twee andere chips mee samenwerken. In het RAM-geheugen slaat de computer allerlei gegevens op: BASIC-programma's, variabelen, de gegevens voor het beeldscherm, en diverse andere gegevens die het systeem nodig heeft.
De grote chip is de ULA (Uncommitted Logic Array)-chip. Dit is het communicatiecentrum van de computer. Deze chip zorgt ervoor dat alle gegevens die de processor wil hebben, daar naar toe gaan en dat de gegevens uit de processor de goede kant op gaan. Daarnaast leest deze chip telkens het geheugen om te zien hoe het beeldscherm eruit moet zien, en stuurt de juiste gegevens naar de TV-uitgang.
De PAL-codeerder is een hele groep componenten die de signalen van de TV-uitgang omzet in een voor een kleurentelevisie geschikt signaal.
De voltageregelaar zet het licht fluctuerende voltage van het lichtnet om in een constante 5 volt spanning.

Hiermee bent u aan het einde van het introduktiedeel van dit boek gekomen. Als u denkt dat u het allemaal door heeft, kunt u verder gaan met deel twee: de BASIC-programmeerhandleiding.

BASIC PROGRAMMEER HANDLEIDING

Inhoud

HOOFDSTUK 1

Introduktie

Of u nu het instruktiedeel van dit boek eerst gelezen heeft, of direkt aan dit deel begonnen bent, met verschillende zaken moet u nu bekend zijn: opdrachten worden direkt uitgevoerd en programmaregels beginnen met een regelnummer. De volgende opdrachten moet u kennen: **PRINT**, **LET** en **INPUT** (deze komen bij alle computers die BASIC gebruiken voor), en **BORDER**, **PAPER** en **BEEP** (welke alleen bij de Spectrum gebruikt worden).

Deze BASIC handleiding begint met het herhalen van enkele zaken die in het eerste deel van dit boek al ter sprake kwamen maar nu veel gedetailleerder; we vertellen u precies wat kan en wat niet kan. U zult aan het einde van elk hoofdstuk een paar opdrachten vinden. Sla deze niet over; veel van de opdrachten laten allerlei zaken zien waar in de tekst op gewezen is. Kijk de opdrachten door en doe die opdrachten die u interessant vindt, of die een aspect bestrijken dat u nog niet helemaal onder de knie heeft.

Wat u ook doet, blijf de computer gebruiken. Als u zich afvraagt "wat zou ie doen als ik hem dit of dat opgeef?" is het antwoord zeer eenvoudig: tik het in en kijk wat er gebeurt. Overal waar deze handleiding u vraagt iets in te tikken moet u zich afvragen "wat had ik anders kunnen intikken?". Tik het in en kijk. Hoe meer programma's u zelf schrijft hoe beter u de computer zult begrijpen.

Aan het einde van deze programmeerhandleiding bevinden zich een paar bijlagen. Daartussen bevindt zich een uitleg van de manier waarop het geheugen georganiseerd is, hoe de computer getallen manipuleert en een stel voorbeeldprogramma's die de mogelijkheden van de Spectrum laten zien.

Het toetsenbord

SPECTRUM karakters bevatten niet alleen de enkele *symbolen* (letters, cijfers, etc.), maar tevens de *tokens*=tekens (sleutelwoorden, funktienamen, etc.) en deze worden alle vanaf het toetsenbord ingevoerd in plaats van geheel uitgespeld te worden. Om al deze funkties en opdrachten te kunnen bevatten hebben sommige toetsen vijf of meer verschillende betekenissen. De verschillende betekenissen krijgt u deels door het gebruiken van de verschillende SHIFT toetsen (bijvoorbeeld het ingedrukt houden van de **CAPS SHIFT** toets of de **SYMBOL SHIFT** toets tijdens het intikken van de gewenste toets) en deels door de computer in verschillende *modi* (= modussen) te laten zijn.

De modus wordt aangegeven door de *cursor*, een knipperende letter die toont waar het volgende karakter van het toetsenbord geplaatst zal worden.

K (van Keywords = sleutelwoord) modus vervangt automatisch de L modus als de

machine een commando of een programmaregel verwacht (in plaats van **INPUT** gegevens), en door de plaats op de regel waar de cursor zich op dat moment bevindt weet de computer of het een regelnummer of een sleutelwoord moet verwachten. Dit gebeurt altijd aan het begin van een regel, net achter **THEN** of net achter **:** (behalve in een string). Als er geen SHIFT toetsen worden gebruikt wordt de eerstvolgende toets geïnterpreteerd als een keyword (op de toetsen) of een cijfer.

L (van Letters) modus is de modus die normaal gesproken, in alle andere gevallen dan hierboven genoemd, in werking is. Als er geen SHIFT toetsen worden gebruikt, wordt het eerstvolgende karakter geïnterpreteerd als het hoofdsymbool op de toets, bij letters de kleine letter.

In zowel K als L modus wordt het gebruik van **SYMBOL SHIFT** en een toets geïnterpreteerd als het kleine rode karakter op de toets en wordt **CAPS SHIFT** met een cijfer geïnterpreteerd als de controle funktie die in wit boven de toets staat. **CAPS SHIFT** met een van de andere toetsen beïnvloedt de sleutelwoorden niet (in K modus). In L modus worden de letters als hoofdletters weergegeven.

C (van Capitals = hoofdletters) modus is een variant van de L modus, waarin alle letters als hoofdletters weergegeven worden. **CAPS LOCK** zorgt voor een verandering van L modus naar C modus of andersom.

E (van Extended = uitgebreide) modus wordt gebruikt om de resterende karakters, meestal tekens, te verkrijgen. Deze modus treedt in werking door allebei de SHIFT toetsen tegelijkertijd in te drukken, en blijft slechts een toetsindruk in werking. In deze modus geeft een lettertoets een karakter of teken (groen boven de toets) bij gebruik zonder SHIFT. Bij gebruik met SHIFT worden het karakter of teken dat in rood onder de toets staat weergegeven. Een cijfertoets geeft een teken als tegelijkertijd de **SYMBOL SHIFT** wordt gebruikt. Anders krijgt u een kleurcontrole karakter.

G (van Graphics = grafiek) modus gaat werken nadat **GRAPHICS** (**CAPS SHIFT** en **9**) ingedrukt is en duurt voort tot dit of **9** opnieuw ingedrukt wordt. Een cijfer geeft een mozaïek grafisch symbool, sla **GRAPHICS** en **DELETE** over, en elke letter behalve V, W, X, Y en Z geven een door de gebruiker gedefinieerd karakter.

Elke toets die langer dan twee of drie seconden ingedrukt wordt, zal zichzelf gaan herhalen.

Invoer vanaf het toetsenbord verschijnt onderaan het scherm, precies zoals het wordt ingetikt. Elk karakter (een enkel symbool of een teken) wordt net voor de cursor ingevoegd. De cursor wordt naar links verplaatst met **CAPS SHIFT** en **5** en naar rechts met **CAPS SHIFT** en **8**. Het laatste karakter voor de cursor kan verwijderd worden met **DELETE** (**CAPS SHIFT** en **0**). (Opmerking: de gehele regel wordt verwijderd door **EDIT** (**CAPS SHIFT** en **1**) en daarna **ENTER**).

Zodra **ENTER** wordt ingedrukt wordt de regel uitgevoerd, in het programma opgeslagen of als **INPUT** gegeven gebruikt naar gelang op dat moment de bedoeling is. Dit gebeurt niet als in de regel een 'syntax error' (vergissing in de grammatika) voorkomt. In dat geval wordt een knipperende **?** vlak naast de vergissing neergezet.

Terwijl programmaregels ingevoerd worden, wordt een listing (weergave van programmaregels) bovenaan het beeld gegeven. De meest recente regel die ingevoerd werd, wordt de *current* (tegenwoordige) regel genoemd en wordt aangegeven door het

symbool ▣ . Dit symbool kan verplaatst worden door gebruik te maken van de toetsen ↓ (**CAPS SHIFT** en **6**) en ↑ (**CAPS SHIFT** en **7**). Als **EDIT** (**CAPS SHIFT** en **1**) ingedrukt wordt, wordt de tegenwoordige regel naar de onderkant van het scherm gebracht en kan daar verbeterd worden.

Als een opdracht uitgevoerd is, of een programma doorlopen, wordt de uitvoer op de bovenste helft van het beeldscherm weergegeven. Dit blijft hier staan totdat **ENTER** getikt wordt met een blanko regel, of ↓ of ↑ wordt ingedrukt. Helemaal beneden aan verschijnt een rapportage door middel van een code (cijfer of letter) die slaat op een mededeling uit bijlage B. Deze rapportage stelt tevens de K modus voor en blijft op het scherm totdat er een toets ingedrukt wordt.

Onder enkele omstandigheden werkt de **CAPS SHIFT** met de **SPACE** toets als een **BREAK** die ervoor zorgt dat de computer stopt en een melding **D** of **L** geeft. Dit gebeurt bij:

a) het einde van een statement terwijl het programma doorlopen wordt, of
b) terwijl de computer de cassetterecorder of de printer gebruikt.

Het televisiescherm

Het scherm heeft 24 regels van elk 32 karakters lang, en wordt verdeeld in twee delen. Het bovenste deel heeft maximaal 22 regels en geeft ofwel een programmalisting ofwel de uitvoer van een programma weer. Als bij het weergeven de onderste regel bereikt is, rolt het hele beeld een regel naar boven. Als dit zou betekenen dat u een regel verliest zonder een redelijke tijd om deze te bekijken, stopt de computer en vraagt **scroll?**. Door de **N**, de **SPACE** of de **STOP** in te drukken stopt het programma met het weergeven van de rapportage **D BREAK - CONT repeats**. Elke andere toets zorgt ervoor dat het rollen (scrollen) van het beeld doorgaat. Het onderste deel van het scherm wordt gebruikt voor het invoeren van opdrachten, programmaregels en **INPUT** gegevens, en voor het weergeven van rapportages. Het onderste deel begint met twee regels, maar breidt zich net zo veel naar boven uit als nodig is voor de hierboven genoemde zaken. Als de onderste helft tekst op de bovenste helft ontmoet, schuift de laatste een regel naar boven op.

HOOFDSTUK 2

BASIC-programmeerconcepten

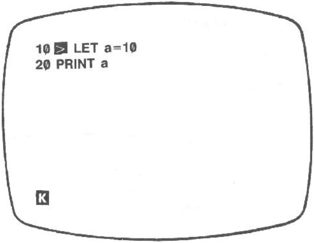
Samenvatting

Programma's
Regelnummers
Redigeren van programma's door ↓, ↑, en **EDIT**
RUN, **LIST**
GO TO, **CONTINUE**, **INPUT**, **NEW**, **REM**, **PRINT**
STOP in **INPUT** gegevens
BREAK

Tik de volgende twee programmaregels in om de som van twee getallen af te drukken:

```
20 PRINT a
10 LET a=10
```

Het scherm moet er nu zo uitzien:



Zoals u al weet, werden deze regels niet onmiddellijk uitgevoerd, maar opgeslagen als programmaregels, omdat ze met getallen beginnen. U heeft ook al opgemerkt dat de regelnummers de volgorde van uitvoeren van programmaregels bepalen. Dit is

belangrijk bij het doorlopen van een programma, maar ook te zien in de volgorde van de regels in de listing zoals die nu op het scherm te zien is.
Tot nu toe heeft u nog maar één getal ingevoerd, dus moet u nu maar tikken

15 LET b=15

en daar wordt de regel tussengevoegd. Het zou niet mogelijk geweest zijn deze regel tussen de eerste twee regels te proppen, als die regels 1 en 2 genummerd waren in plaats van 1Ø en 2Ø (regelnummers moeten gehele getallen tussen 1 en 9999 zijn). daarom moet u zich vanaf het eerste programmeerbegin eigen maken om voldoende ruimte te laten tussen de verschillende regelnummers.
Regel 2Ø moet nu aangepast worden tot

2Ø PRINT a+b

U zou de vervanging natuurlijk helemaal kunnen uittikken, maar het is veel gemakkelijker om de **EDIT** faciliteit te gebruiken die in het eerste deel van dit boek behandeld is. De ▶ in regel 15 wordt de programmacursor genoemd. De regel waarnaar deze wijst is de tegenwoordige regel (current line). Dit is normaal gesproken de laatste door u ingetikte regel, maar u kunt de ↑ of ↓ toetsen gebruiken om de programmaregel van boven naar beneden en vice versa te verplaatsen. (Probeer dit uit en laat de cursor uiteindelijk bij regel 2Ø uitkomen.)
Als u nu de **EDIT** toets indrukt, verschijnt er een kopie van de tegenwoordige regel onderaan het scherm. In dit geval een kopie van regel 2Ø. Hou de → toets vast totdat de **L** cursor naar het einde van de regel beweegt en tik dan

+b (zonder **ENTER**)

De regel onderaan het scherm zou er nu zo uit moeten zien:

2Ø PRINT a+b

Tik **ENTER** en de oude regel 2Ø zal vervangen worden, zodat het scherm er nu zo uit ziet:

```
1Ø  LET a=1Ø
15  LET b=15
2Ø▶ PRINT a+b



K
```

Run dit programma (laat dit programma uitvoeren), door **RUN** en **ENTER** en de som zal weergegeven worden.
Run het programma opnieuw en tik dan

PRINT a,b

De variabelen zijn er nog steeds, hoewel het programma al afgelopen is.
Er is een bruikbare methode met behulp van **EDIT** om van de onderkant van het scherm af te komen. Tik een heleboel onzin in (zonder **ENTER**), en beslis dan dat u dit toch maar niet wilt gebruiken. Een manier om het weg te krijgen is een tijdje de **DELETE** toets ingedrukt houden totdat alles verwijderd is. Maar er is ook een snellere methode. Als u **EDIT** tikt, wordt de onzin onderaan het scherm vervangen door een kopie van de tegenwoordige regel. Als u nu **ENTER** intikt, wordt de tegenwoordige regel onveranderd terug gezet in het programma, waarbij de onderkant van het scherm leeg blijft.
Als u een regel per ongeluk invoert, bijvoorbeeld

12 LET b=8

gaat deze naar boven het programma in, en dan ondekt u uw fout te laat. Om deze onnodige regel te verwijderen tikt u

12 (met **ENTER** natuurlijk)

U zult met verbazing merken dat de programmacursor verdwenen is. U moet zich voorstellen dat deze verborgen is ergens tussen regel 1Ø en regel 15. Als u ↑ tikt gaat de programmacursor een regel omhoog en komt tevoorschijn voor regel 1Ø, en als u ↓ tikt komt hij tevoorschijn voor regel 15.
Tik

12 (en **ENTER**)

De programmacursor zal opnieuw verdwijnen tussen regel 1Ø en 15. Druk nu op **EDIT** en regel 15 zal naar beneden komen: Als de programmacursor tussen twee regels in verborgen is, brengt **EDIT** de regel met het regelnummer dat volgt op de regel waar de programmacursor verdween naar beneden. Tik **ENTER** om het onderste deel van het scherm schoon te krijgen.
Tik nu

3Ø (en **ENTER**)

Deze keer is de programmacursor verborgen achter het einde van het programma: Als u **EDIT** indrukt, komt regel 2Ø naar beneden.
Tik tenslotte

LIST 15

U ziet nu op het scherm

```
15 LET b=15
20 PRINT a+b
```

Regel 10 is van uw scherm verdwenen, maar zit nog wel in uw programma. U kunt dit bewijzen door **ENTER** in te drukken. De enige effecten van **LIST 15** zijn het produceren van een listing die begint op regel 15, en het zetten van de programmacursor op regel 15. Als u een erg lange programmalisting heeft is het verplaatsen van de programmalisting met behulp van **LIST** waarschijnlijk sneller dan door middel van ↓ en ↑.
Dit illustreert nog een ander aspect aan regelnummers; ze dienen als naam voor programmaregels, zodat er naar een bepaalde programmaregel verwezen kan worden. Dit lijkt op het hebben van namen door variabelen.
LIST alleen geeft een listing die bij het begin van het programma begint.
Een andere opdracht die in deel een van dit boek staat is

NEW

Deze opdracht wist elk oud programma en de variabelen die in de computer aanwezig zijn. Tik voorzichtig het volgende programma, dat Fahrenheit temperaturen omzet in Celcius weergave, in.

```
10 REM temperatuur omzetting
20 PRINT "graden F","graden C"
30 PRINT
40 INPUT "Voer graden F in",F
50 PRINT F,(F-32)*5/9
60 GO TO 40
```

De woorden in regel 10 moeten letter voor letter ingetikt worden. De opdracht **GO TO** heeft weliswaar een spatie in het midden, maar is toch een sleutelwoord (op de **G**). Run het programma nu. U ziet het kopje door regel 20 op het scherm afgedrukt worden. Maar wat is ergebeurdmet regel 10? Schijnbaar heeft de computer deze regel helemaal genegeerd. Klopt, dat heeft hij ook. **REM** in regel 10 staat voor remark (opmerking) of reminder (herinnering), en dient alleen om u eraan te herinneren wat het programma doet. Een **REM** opdracht bestaat uit een **REM** gevolgd door wat u maar wilt. De computer zal alles achter de **REM** negeren, tot het einde van de regel.
Ondertussen is de computer blijven wachten bij de **INPUT** opdracht van regel 40 totdat u een waarde voor de variabele F indrukt. U kunt dit zien aan de cursor. Waar u een **K** cursor verwachtte staat een knipperende **L**. Voer een getal in; vergeet **ENTER** niet. Nu geeft de computer het resultaat weer en wacht op het volgende getal. Dit komt

door regel 6Ø **GO TO 4Ø** die precies betekent wat de regel zegt (in het Engels dan wel): ga naar 4Ø. In plaats van aan het eind van het programma te komen en te stoppen, springt de computer terug naar regel 4Ø en begint daar opnieuw. Tik dus nog een temperatuur in.
Na dit een paar keer gedaan te hebben, vraagt u zich misschien af of de computer hier nooit genoeg van krijgt. Nee, de computer krijgt hier nooit genoeg van. De volgende keer dat hij weer om een getal vraagt kunt u **STOP** intikken. De computer antwoordt dan met de rapportage **H STOP in INPUT in line 4Ø:1** hetgeen u zegt waarom hij stopte, en waar (de eerste opdracht van regel 4Ø).
Als u door wilt gaan met het programma tikt u

CONTINUE

en de computer vraagt om nog een getal.
Als **CONTINUE** (ga door, continueer) wordt gebruikt, onthoudt de computer het getal van de laatste rapportage die naar u gezonden werd, zolang dat geen **Ø OK** was, en springt terug naar die regel. In dit voorbeeld was dat naar regel 4Ø springen, de **INPUT** opdracht.
Vervang regel 6Ø door **GO TO 31** en het maakt voor het doorlopen van het programma geen merkbaar verschil. Als het regelnummer in een **GO TO** opdracht naar een niet bestaande regel verwijst, springt de computer naar de eerstvolgende wel bestaande regel. hetzelfde geldt voor **RUN**. In feite betekent **RUN** alleen eigenlijk **RUN Ø**. Tik nu net zo lang getallen in todat het scherm vol begint te raken. Als het scherm vol is, zal de computer het gehele bovenste gedeelte een regel naar boven schuiven om ruimte te maken. De bovenste regel gaat verloren. Dit heet scrolling of rollen.
Als u hier genoeg van heeft stopt u het programma met **STOP** en haalt u de listing tevoorschijn met behulp van **ENTER**.
Kijk goed naar de **PRINT** opdracht van regel 5Ø. De punktuering - de komma (,) - is heel belangrijk, en u moet onthouden dat deze veel striktere regels volgt dan de punktuering in het Nederlands of Engels.
Komma's worden gebruikt om het afdrukken te laten beginnen aan de linker kantlijn, of in het midden van het scherm, naar gelang wat er achter aan komt. In regel 5Ø zorgt de komma ervoor dat de Celcius graden in het midden van de regel wordt afgedrukt. Met een punt-komma echter wordt het volgende getal of de volgende string direkt achter de voorgaande geplaatst. U kunt dit zien in regel 5Ø, als de komma vervangen wordt door een punt-komma.
Een ander leesteken dat u op deze wijze kunt gebruiken bij **PRINT** opdrachten is het enkele aanhalingsteken ('). Dit zorgt ervoor dat de af te drukken tekst op de eerstvolgende nieuwe regel, bij de linker kantlijn geplaatst wordt. Dit gebeurt ook bij elke nieuwe **PRINT** opdracht, dus u heeft dit leesteken waarschijnlijk niet zo vaak nodig. Daarom ook begint de **PRINT** opdracht van regel 5Ø, de nieuwe tekst telkens op een nieuwe regel, en daarom ook geeft de **PRINT** opdracht van regel 3Ø telkens een blanko regel.
Als u dit wilt voorkomen, zodat achter de tekst van de ene **PRINT** opdracht, de tekst

van de volgende **PRINT** opdracht aansluitend geplaatst wordt, moet u een punt-komma achter de eerste opdracht zetten. (Of, als u een half beeldscherm tussenruimte wilt, een komma). Om te zien hoe dit werkt kunt u regel 5Ø een paar keer veranderen:

5Ø PRINT F,
5Ø PRINT F;

en

5Ø PRINT F

en run elke versie. Voor de goede orde kunt u ook proberen

5Ø PRINT F'

De versie met de komma verdeelt alles in twee kolommen, terwijl de versie met de punt-komma alles bij elkaar propt. De versie zonder iets geeft voor elk getal een nieuwe regel, net als de versie met het aanhalingsteken. Het aanhalingsteken geeft van zichzelf een nieuwe regel, maar onderdrukt de nieuwe regel die automatisch optreedt. Vergeet dit verschil tussen de verschillende leestekens in **PRINT** opdrachten niet. Verwissel ze ook niet met de dubbele punt (:), die gebruikt wordt om opdrachten op een regel van elkaar te scheiden.
Tik nu de volgende extra regels in:

```
100 REM Dit beleefde programma onthoudt uw naam
110 INPUT n$
120 PRINT "Hallo ";n$;"!"
130 GO TO 110
```

Dit programma staat los van het vorige, maar u kunt ze allebei tegelijkertijd in de computer opslaan. Om het nieuwe programma te runnen tikt u

RUN 1ØØ

Omdat dit programma een string als invoer wil hebben, in plaats van een getal, drukt het twee aanhalingstekens voor een string af. Dit is een herinnering voor u, en het bespaart u meestal nogal wat tikwerk. Probeer het maar eens met een willekeurig pseudoniem voor uzelf.
De volgende ronde krijgt u weer twee string aanhalingstekens. U bent echter niet verplicht ze te gebruiken. Probeer dit bijvoorbeeld eens. Wis de aanhalingstekens uit (met → en **DELETE** twee keer), en tik

n$

Omdat er geen string aanhalingstekens zijn, weet de computer dat er gerekend moet worden. Het rekenwerk is in dit geval het vinden van de waarde van de stringvariabele n$, hetgeen datgene is wat u in de laatste ronde had ingetikt. Natuurlijk werkt de

INPUT opdracht als **LET n$=n$**, dus de waarde van n$ blijft onveranderd. De volgende ronde kunt u bijvoorbeeld ter vergelijking opnieuw

n$

tikken. Dit keer zonder de aanhalingstekens uit te wissen. Nu heeft de variabele n$, om u te verwarren, de waarde "n$".
Als u bij het invoeren van strings **STOP** wilt gebruiken, moet u de cursor eerst naar het begin van de regel bewegen door middel van ←.
Kijk nu nog eens naar dat **RUN 100** van hierboven. Door deze opdracht springt de computer naar regel 100. Kan daarvoor dan niet net zo goed **GO TO 100** gebruikt worden? In dit speciale geval is het antwoord ja, maar er is wel een verschil tussen de twee opdrachten. **RUN 100** zorgt er eerst voor dat alle variabelen schoon gemaakt zijn, en dat het scherm gewist is, pas daarna werkt het precies als **GO TO 100**. **GO TO 100** wist helemaal niets, het kan best voorkomen dat u een programma wilt runnen zonder dat de variabelen gewist worden. In zo'n geval moet u **GO TO** gebruiken, omdat **RUN** een heel ander gevolg heeft. Verval daarom niet in een automatisme om altijd **RUN** te gebruiken.
Een ander verschil is dat u **RUN** ook zonder regelnummer kunt gebruiken, waardoor begonnen wordt bij de eerste regel. **GO TO** moet altijd een regelnummer hebben.
Allebei de vorige programma's stopten omdat u een **STOP** intikte bij de invoer regel.
Het komt soms voor dat u -per ongeluk- een programma schrijft dat u niet kunt stoppen, en dat zichzelf ook niet stopt. Tik

```
200 GO TO 200
RUN 200
```

Dit ziet er uit alsof het programma altijd zal blijven lopen, tenzij de stekker uit het stopkontakt gehaald wordt. Er is een minder drastische methode. Tik **CAPS SHIFT** met de **SPACE** toets, waarboven **BREAK** staat. Het programma zal dan stoppen en als rapportage geven **L BREAK into program, 200:1**.
Na elke opdracht bekijk het programma of deze toetsen ingedrukt zijn, en als ze dat zijn, stopt het programma. De **BREAK** toets kan ook gebruikt worden als u de cassetterecorder of de printer aan het gebruiken bent, of andere randpparatuur, in het geval dat de computer op invoer van een van deze apparaten wacht, maar deze apparaten het om de een of andere reden niet doen.
U krijgt in deze gevallen wel een andere rapportage: **D BREAK - CONT repeats**. Als u in dit geval (maar ook in de meeste andere gevallen) **CONTINUE** gebruikt, gaat de computer verder met de opdracht waarmee hij bezig was toen het programma onderbroken werd, waarbij deze laatste opdracht opnieuw gedaan wordt. Na een **L BREAK into program**, zorgt **CONTINUE** ervoor dat naar de opdracht volgend op die waarmee het programma bezig was tijdens de onderbreking, springt.
Run het programma nog een keer en tik dit keer

n$ (na de aanhalingstekens weggehaald te hebben)

n$ is een niet gedefinieerde variabele en u krijgt een fout-rapportage **2 Variabele not found**. Als u nu tikt

LET n$="iets gedefinieerds"

(welke opdracht een eigen rapportage **Ø OK, Ø:1** heeft) en

CONTINUE

ziet u dat u zonder enig bezwaar n$ als invoergegeven kunt gebruiken.
In dit geval springt **CONTINUE** naar de **INPUT** opdracht in regel 11Ø. De rapportage van de **LET** opdracht wordt genegeerd, want die was 'OK'. De sprong gaat naar de opdracht waarnaar in de vorige rapportage verwezen was, de eerste opdracht in regel 11Ø. Dit is bedoeld als handig. Als een programma stopt vanwege een fout, kunt u van alles doen om de fout te repareren, terwijl **CONTINUE** daarna toch nog werkt.
Zoals we al eerder zeiden is de rapportge **L BREAK into program** speciaal, omdat hierna de **CONTINUE** niet de opdracht waar het programma stopte herhaalt.
De automatische listings (de listings die niet het resultaat zijn van een **LIST** opdracht, maar die telkens voorkomen na het invoeren van een nieuwe regel), zijn wellicht raadselachtig. Als u een programma met 5Ø regels intikt, elk alleen bestaande uit een **REM** opdracht, kunt u experimenteren.

```
1 REM
2 REM
3 REM
: :
: :
49 REM
5Ø REM
```

Het eerste dat u zich moet herinneren is dat de current, tegenwoordige regel (met ▶) altijd op het scherm staat, meestal in het midden.
Tik

LIST (en **ENTER** natuurlijk)

en als de computer vraagt **scroll?** (rollen?), omdat het scherm vol is, tikt u **n** van 'nee'. De computer geeft dan de rapportage **D BREAK - CONT repeats** alsof u **BREAK** getikt had. U zult nog ontdekken wat er gebeurt als u **y** (van 'yes' = ja) tikt in plaats van **n**. **n**, **SPACE** en **STOP** tellen als nee, alle andere toetsen tellen als ja (**y**).
Druk nu op **ENTER** om een automatische listing te krijgen. U krijgt daardoor regel 1 tot en met 22 op het scherm. Tik nu

23 REM

en u krijgt de regels 2 tot en met 23 op het scherm. Tik

28 REM

en u krijgt de regel 7 tot en met 28. In beide gevallen heeft u door een nieuwe regel te tikken de programma cursor verplaatst zodat een nieuw listing gemaakt wordt.
Misschien lijkt dit alles een beetje arbitrair. In feite probeert de computer u precies te geven wat u wilt, maar omdat mensen nu eenmaal onvoorspelbare wezens zijn, gokt de computer niet altijd goed over uw wensen.
Als de bovenste regel toch wegvalt, is het zinloos om daar te beginnen, dus wordt de tegenwoordige regel (current) gebruikt als bovenste regel en wordt er vanaf daar gelist.
Verder is de methode om de listing te maken vanaf de bovenste regel, en door te gaan tot de tegenwoordige regel, waarbij eventueel het beeld gaat scrollen. Maar de computer doet even een ruwe berekening hoe lang dit gaat duren, en als dit veel te lang is, wordt de bovenste regel een stuk richting tegenwoordige regel verschoven, en wordt daar begonnen met listen. Als op deze manier bepaald is wat de bovenste regel is, wordt daar begonnen met listen. Als het einde van het programma of de bodem van het scherm bereikt is, en de tegenwoordige regel is gelist, stopt hij. Anders gaat het beeld scrollen, net zo lang tot de tegenwoordige regel gelist is. Daarbij wordt telkens een opgeteld bij de bovenste regel, zodat deze altijd in de buurt van de tegenwoordige regel blijft.
Experimenteer met het verplaatsen van de tegenwoordige regel door te tikken

regelnummer **REM**

LIST verplaatst de tegenwoordige regel maar niet de bovenste regel, dus de volgende listings kunnen er verschillend uit zien. Tik

LIST

om de **LIST** listing te krijgen, en tik **ENTER** om Ø weer de bovenste regel te maken. U moet dan regel 1 tot en met 22 op het scherm hebben. Tik

LIST 22

zodat u regel 22 tot en met 43 krijgt. Door opnieuw **ENTER** in te drukken krijgt u regel 1 tot en met 22 weer. Dit is handiger bij kleine listings dan bij lange.

Gebruik het programma met alleen **REM**s hierboven en tik

LIST

en dan **n** als de vraag komt **scroll?**. Tik nu

CONTINUE

CONTINUE is hier een beetje raar, omdat het onderste deel van het scherm helemaal leeg wordt. Maar u kunt weer terug komen met **BREAK**. De reden hiervoor is dat **LIST** de eerste opdracht op de regel was, dus **CONTINUE** herhaalt deze opdracht. Maar de eerste opdracht in de regel is nu **CONTINUE** zelf, zodat de computer niets anders doet dan **CONTINUE** de hele tijd, totdat u dat stopt.
U kunt dit varieren door **LIST** te vervangen door

:LIST

waarbij **CONTINUE** als melding **Ø OK** geeft (omdat **CONTINUE** naar de tweede opdracht in de regel springt, die tevens het einde is) of

::LIST

waarbij **CONTINUE** als melding **N Statement lost** (=N opdracht verloren) geeft omdat **CONTINUE** naar de derde opdracht springt, die niet meer bestaat.
U heeft nu de opdrachten **PRINT, LET, INPUT, RUN, LIST, GO TO, CONTINUE, NEW** en **REM** gezien, en deze kunnen allen gebruikt worden als direkte opdrachten of in programmaregels. Bijna alle opdrachten in ZX Spectrum BASIC kunnen op beide manieren gebruikt worden. **RUN, LIST, CONTINUE** en **NEW** zijn meestal niet nuttig in een programma, maar ze kunnen er wel in gebruikt worden.

Oefeningen

1. Stop een **LIST** opdracht in een programma, zodat als het programma gerund wordt, het zichzelf list.

2. Schrijf een programma om prijzen in te voeren en de BTW uit te rekenen (19 %). Gebruik **PRINT** opdrachten zodat het duidelijk is wat de computer gaat doen, en zorg dat de computer heel beleefd om prijzen als invoer gegevens vraagt. Verander het programma vervolgens zo, dat de BTW tarieven te veranderen zijn.

3. Schrijf een programma dat telkens de som van alle tot dan toe ingevoerde getallen weergeeft. (suggestie: gebruik twee variabelen met de namen **totaal** -in het begin gelijk aan Ø- en **item**. Voer **item** telkens in, tel het op bij **totaal**, druk ze allebei af, en begin weer bij het begin.)

4. Wat zouden **CONTINUE** en **NEW** in een programma doen? Kunt u iets nuttigs verzinnen voor het gebruik van deze twee in een programma?

HOOFDSTUK 3

Beslissingen

Samenvatting

IF, STOP
=, <, >, < =, > =, < >

Alle programma's die we tot nu toe hebben gezien waren nogal voorspelbaar. Ze gingen rechtstreeks door alle opdrachten, en gingen daarna weer naar het begin. Dit is niet erg bruikbaar. In de praktijk is het nodig dat de computer beslissingen neemt en aan de hand van die beslissingen handelt. De opdracht die daarvoor gebruikt wordt heeft de vorm: **IF** (ALS) iets (de conditie) waar is, of niet-waar is, **THEN** (DAN) doe iets. Bijvoorbeeld: gebruik **NEW** om het vorige programma te wissen en tik het volgende programma in en run het. (Het is bedoeld voor twee spelers!)

```
10 REM raad het getal
20 INPUT a: CLS
30 INPUT "Raad het getal",b
40 IF b=a THEN PRINT "Juist":STOP
50 IF b < a THEN PRINT "Te klein, probeer nog eens"
60 IF b > a THEN PRINT "Te groot, probeer nog eens"
70 GO TO 30
```

U ziet dat een **IF** opdracht als vorm heeft:

IF voorwaarde **THEN** . . .

waarbij de '. . .' staat voor voor een stel opdrachten die door middel van dubbele punten zijn gescheiden. De voorwaarde is iets dat uitgewerkt wordt en dan of waar of niet-waar is: Als het waar is, worden de opdrachten achter **THEN** op de regel uitgevoerd, anders worden deze opdrachten overgeslagen en gaat het programma verder met de volgende regel.
De eenvoudigste condities vergelijken twee getallen of twee strings. Zij kunnen bekijken of twee getallen gelijk zijn, of dat een van de twee groter is. Bij strings kunnen zij kijken of twee strings gelijk zijn, of dat een van de twee (ruwweg) voor de ander komt in alfabetische volgorde. De vergelijkingstekens die gebruikt worden zijn: =, <, >, < =, > = en < > = betekent 'is gelijk'. Hoewel het hetzelfde symbool is als de = in de **LET** opdracht wordt het toch op een heel andere manier gebruikt.

< (**SYMBOL SHIFT** met **R**) betekent 'is minder dan', dus

1 < 2
-2 < -1
-3 < -1

zijn allemaal waar, maar

1 < Ø
Ø < -2

zijn allebei niet-waar.
Regel 4Ø vergelijkt **a** en **b**. Als zij gelijk zijn, dan wordt het programma gestopt door de **STOP** opdracht. De rapportage aan de onderkant van het scherm zegt dan **9 STOP, statement, 3Ø:3** waaruit blijkt dat de computer gestopt is vanwege een **STOP** opdracht (statement) in het derde deel van regel 3Ø.
Regel 5Ø bepaalt of **b** kleiner is dan **a**, en regel 6Ø bekijkt of **b** groter is dan **a**. Als een van deze twee voorwaarden waar is, wordt de toepasselijke tekst afgedrukt, en het programma loopt door tot regel 7Ø, waar de computer teruggestuurd wordt naar regel 3Ø om nogmaals te beginnen.
De **CLS** opdracht (CLear Screen = wis scherm) in regel 2Ø zorgde ervoor dat de ander niet kon zien wat u intikte.
Dus > (**SYMBOL SHIFT** met **T**) betekent 'is groter dan', en is hetzelfde als < maar dan de andere kant op. U kunt dit onthouden doordat het puntje altijd wijst naar de waarde die geacht wordt kleiner te zijn.
< = (**SYMBOL SHIFT** met **Q** - niet intikken als < met daarna een =) betekent 'is kleiner dan of gelijk aan', en is dus gelijk aan < behalve dat het ook waar is als de twee waarden gelijk zijn. 2 < =2 is waar, 2 < 2 is niet-waar.
> = (**SYMBOL SHIFT** met **E**) betekent 'is groter dan of gelijk aan' en is vergelijkbaar met >.
< > (**SYMBOL SHIFT** met **W**) betekent 'is niet gelijk aan', en is het omgekeerde van =.
Wiskundigen schrijven normaal gesproken < =, > = en < > als,
≮, ≯ en ≠. Ze schrijven ook wel eens zaken als '2 < 3 < 4' om te zeggen '2 < 3 en 3 < 4', maar dit is niet mogelijk in BASIC.
Merk op dat in sommige BASIC versies - maar *niet* op de ZX Spectrum - de **IF** opdracht er uit kan zien als

IF voorwaarde THEN regelnummer

Dit betekent hetzelfde als

IF conditie **THEN GO TO** regelnummer.

Oefeningen

1. Probeer dit programma

```
10 PRINT "x": STOP: PRINT "y"
```

Als u het laat lopen, geeft het een **x** weer en stopt het met de melding **9 STOP statement; 10:2**.
Tik nu

CONTINUE

U zou nu verwachten dat door deze opdracht teruggesprongen wordt naar de **STOP** opdracht omdat **CONTINUE** meestal de opdracht waarnaar verwezen wordt in de rapportage herhaald. Dit zou echter niet zo erg nuttig zijn, daar de computer opnieuw zou stoppen, nog steeds zonder **y** weergegeven te hebben. Daarom is de computer intern zo geregeld dat na een rapportage nummer 9, **CONTINUE** naar de opdracht na de **STOP** opdracht springt. In ons voorbeeld drukt de computer na de **CONTINUE** de letter **y** af en bereikt het einde van het programma.

HOOFDSTUK 4

Lussen

Samenvatting
FOR, NEXT
TO, STEP

Stel u wilt 5 getallen invoeren en hen optellen. Een van de methoden om dat te bereiken is met het volgende programma (niet intikken, tenzij u overdreven plichtsbesef heeft):

```
 10 LET totaal=0
 20 INPUT a
 30 LET totaal=totaal+a
 40 INPUT a
 50 LET totaal=totaal+a
 60 INPUT a
 70 LET totaal=totaal+a
 80 INPUT a
 90 LET totaal=totaal+a
100 INPUT a
110 LET totaal=totaal+a
120 PRINT totaal
```

Van deze methode kan niet gezegd worden dat het een goede manier van programmeren is. Het is misschien nog net allemaal controleerbaar voor 5 getallen, maar u kunt zich voorstellen hoe vervelend het wordt om 10 getallen in te tikken, en een programma om 100 getallen op te tellen is bijna onmogelijk.
Veel beter is het om een variabele op te stellen, die tot 5 telt en daarna het programma stopt, zoals in het volgende programma (dat u wel kunt intikken)

```
10 LET totaal=0
20 LET teller=1
30 INPUT a
40 REM teller=aantal keren dat er tot nu toe invoer geweest is
50 LET totaal=totaal+a
60 LET teller=teller+1
70 IF teller <=5 THEN GO TO 30
80 PRINT totaal
```

Merk op hoe eenvoudig het is om regel 70 te veranderen zodat dit programma 10 ge-

tallen optelt, of zelfs 1ØØ.
Deze manier van tellen is zo handig dat er twee speciale opdrachten gemaakt zijn om het nog eenvoudiger te maken: de **FOR** opdracht (voor) en de **NEXT** opdracht (volgende). Deze twee opdrachten worden *altijd* samen gebruikt.
Als deze twee opdrachten waren gebruikt had het vorige programma er zo uitgezien:

```
10 LET totaal=0
20 FOR c=1 TO 5
30 INPUT a
40 REM c=aantal keren dat er tot nu toe invoer geweest is
50 LET totaal=totaal+a
60 NEXT c
80 PRINT totaal
```

(om dit programma uit het vorige te krijgen moet u de regels 2Ø, 4Ø, 6Ø en 7Ø veranderen. **TO** is **SYMBOL SHIFT** met **F**).
Merk op dat we **teller** hebben veranderd in **c** (van count). De teller variabele -of *controle* variabele- van een **FOR..NEXT** lus moet een enkele letter als naam hebben.
Het effect van dit programma is dat **c** door de waarden 1 (de *initiële*=beginwaarde), 2, 3, 4 en 5 (de *limiet*) loopt, en voor elke waarde worden de regels 3Ø, 4Ø en 5Ø uitgevoerd. Als **c** dan zijn 5 waarden heeft gehad, wordt regel 8Ø uitgevoerd.
Een extra subtiliteit bij dit alles is dat de controle variabele niet telkens met 1 omhoog hoeft. U kunt deze 1 in wat u maar wilt veranderen door een **STEP** onderdeel in de **FOR** opdracht op te nemen. De meest algemene vorm voor een **FOR** opdracht is

FOR controle variabele = initiële waarde **TO** limiet **STEP** stapgrootte

waarbij de controle variabele een enkele letter is, en de initiële waarde, de limiet en de stapgrootte allemaal zaken zijn die de computer als getallen kan uitrekenen, zoals getallen zelf, sommen, of de namen van numerieke variabelen (variabelen die alleen getallen als waarden kunnen hebben). Dus als u in regel 2Ø het programma verandert tot

```
20 FOR c=1 to 5 STEP 3/2
```

zal **c** de waarden 1, 2.5 en 4 krijgen. Merk op dat u zich niet hoeft te beperken tot gehele getallen, en dat de controle waarde de limiet niet precies hoeft te bereiken. De lus gaat door zolang de controle variabele kleiner of gelijk aan de limiet is.
Probeer dit programma dat de getallen van 1 tot en met 1Ø in omgekeerde volgorde weergeeft.

```
10 FOR n=10 TO 1 STEP -1
20 PRINT n
30 NEXT n
```

We zeiden al eerder dat het programma blijft lopen zolang de controle variabele gelijk aan of kleiner dan de limiet is. Als u dat uitwerkt voor bovenstaand voorbeeld blijkt dat dat in dit geval nonsens zou zijn. De normale regel moet dus aangepast worden. Als de stapgrootte negatief is, blijft de programmalus lopen zolang de controlevariabele groter of gelijk aan de limiet is.
U moet voorzichtig zijn als u twee **FOR..NEXT** lussen tegelijkertijd gebruikt, waarbij een zich binnen de ander bevindt. Probeer dit programma dat alle getallen van een compleet dominospel weergeeft.

```
10 FOR m=0 TO 6                          ⎤
20 FOR n=0 TO m          ⎫               ⎥
30 PRINT m;":";n;" ";    ⎬  n-lus        ⎬  m-lus
40 NEXT n                ⎭               ⎥
50 PRINT                                 ⎥
60 NEXT m                                ⎦
```

U ziet dat de **n**-lus volledig binnen de **m**-lus ligt. Zij zijn *goed genesteld.* Wat vermeden moet worden is dat twee **FOR..NEXT** lussen overlappen zonder dat ze volledig binnen elkaar liggen zoals bij dit programma

```
 5 REM dit programma is fout
10 FOR m=0 TO 6          ⎫
20 FOR n=0 TO m          ⎬  m-lus    ⎫
30 PRINT m;":";n;" ";    ⎪           ⎪
40 NEXT m                ⎭           ⎬  n-lus
50 PRINT                             ⎪
60 NEXT n                            ⎭
```

Twee **FOR..NEXT** lussen moeten ofwel helemaal binnen elkaar, ofwel helemaal apart zijn.
Nog iets om te voorkomen is midden in een **FOR..NEXT** lus springen van buiten af. De controle variabele wordt alleen goed neergezet als eerst de **FOR** opdracht uitgevoerd is. Als deze gemist wordt, raakt de computer in verwarring bij de eerstvolgende **NEXT** opdracht. U krijgt dan waarschijnlijk een foutmelding die zegt **NEXT without FOR** (NEXT zonder FOR) of **variable not found** (variabele niet gevonden).
Er is niets dat u tegenhoudt om **FOR** en **NEXT** in direkte opdrachten te gebruiken. Probeer bijvoorbeeld

```
FOR m=0 TO 10: PRINT m: NEXT m
```

U kunt dit soms gebruiken als een (wat kunstmatige) manier om van de restrictie af te komen dat een **GO TO** niet binnen een opdracht gebruikt kan worden, omdat een opdracht geen regelnummer heeft. Bijvoorbeeld:

```
FOR m=0 TO 1 STEP 0: INPUT a: PRINT a : NEXT m
```

De stapgrootte nul zorgt er hier voor dat de opdracht zichzelf altijd blijft herhalen. Dit soort methodes zijn niet echt raadzaam. Als er ergens een vergissing in zit, bent u de opdracht kwijt, en moet u hem opnieuw intikken. **CONTINUE** zal hierbij niet helpen.

Oefeningen

1. Een controle variabele heeft niet alleen een naam en een waarde zoals elke variabele, maar ook een limiet, een stapgrootte en een referentie naar de opdracht achter de bijhorende **FOR** opdracht. Overtuig uzelf dat zodra de **FOR** opdracht uitgevoerd is, al deze informatie ook aanwezig is (door gebruik te maken van de initiële waarde als de eerste waarde die de variabele aanneemt). Controleer ook dat deze informatie voldoende is voor de volgende **NEXT** opdracht om te weten met hoeveel de waarde verhoogd moet worden, of er terug gesprongen moet worden, en zo ja, waar naar toe gesprongen moet worden.

2. Run het derde programma hierboven en tik dan

PRINT c

Waarom is het antwoord 6, en niet 5?
(Antwoord: de **NEXT** opdracht in regel 6Ø wordt 5 keer uitgevoerd, en elke keer wordt er 1 opgeteld bij **c**. De laatste keer wordt **c** dus 6; dan beslist de **NEXT** opdracht om niet terug te springen maar door te gaan, omdat **c** voorbij de limiet is.)
Wat gebeurt er als u **STEP 2** in regel 2Ø zet?

3. Verander het derde programma zodanig dat u gevraagd wordt hoeveel getallen u in wilt vullen, zodat er niet automatisch 5 getallen genomen worden. Als u dit programma runt en u voert Ø in om aan te geven dat u helemaal geen getallen wilt invoeren, wat gebeurt er dan? Waarom zou u problemen voor de computer kunnen verwachten terwijl toch duidelijk is wat u bedoelt? (De computer moet op zoek naar de **NEXT c** opdracht, hetgeen niet altijd nodig is). In feite is hier al aan gedacht.

4. In regel 4Ø van het vierde programma hierboven, kunt u **1Ø in 1ØØ** veranderen. U krijgt dan de getallen van 1ØØ tot en met 79 op het scherm en de vraag **scroll?** aan de onderkant. Dit geeft u de mogelijkheid de getallen die aan de bovenkant van het scherm gaan lopen te bekijken. Als u **n** tikt, of **STOP** of **BREAK**, stopt het programma met de melding **D BREAK - CONT repeats**. Als u een willekeurige andere toets indrukt, worden er weer 22 regels afgedrukt, en herhaalt de vraag zich.

5. Verwijder regel 3Ø uit het vierde programma. Als u dit ingekorte programma runt, wordt het eerste getal weergegeven en stopt het programma met de melding **Ø OK**. Als u tikt

NEXT n

gaat het programma een keer door de lus, en drukt het volgende getal af.

HOOFDSTUK 5

Subroutines

Samenvatting
GO SUB, RETURN

Soms hebben verschillende delen van het programma ongeveer gelijke taken te doen, en merkt u dat u dezelfde regel twee of meer keer aan het intikken bent. Dit is echter niet echt noodzakelijk. U kunt de regels een keer intikken, in een speciale vorm die subroutine heet, en deze dan telkens opnieuw gebruiken, of aanroepen, in verschillende delen van het programma. U hoeft ze daardoor dan geen tweede keer in te tikken. Om dit te doen gebruikt u de opdrachten **GO SUB** (**GO** to **SUB**routine = ga naar subroutine) en **RETURN** (= keer terug).
De vorm hiervoor is

GO SUB n

waarbij n het regelnummer van de eerste regel in de subroutine is. Het is net als **GO TO** n, met als enige verschil dat de computer onthoudt waar de **GO SUB** opdracht stond, zodat hij later het programma voort kan zetten bij de regel waar hij gebleven was voordat hij de subroutine moest doen. De computer doet dit door het regelnummer, en het opdrachtnummer (binnen de regel) - deze vormen samen het *terugkeer adres* - te nemen en deze op een speciale stapel (de **GO SUB** stapel) te plaatsen;

RETURN

neemt het terugkeer adres van de **GO SUB** stapel af, en gaat naar de opdracht die er onmiddellijk op volgt.
Als voorbeeld kijken we naar het raad-het-getal-programma. Tik het opnieuw volgens onderstaand voorbeeld in.

```
 10 REM "Veranderd raad spel"
 20 INPUT a: CLS
 30 INPUT "Raad het getal ",b
 40 IF a=b THEN PRINT "Goed":STOP
 50 IF a<b THEN GO SUB 100
 60 IF a>b THEN GO SUB 100
 70 GO TO 30
100 PRINT "Probeer nog eens"
110 RETURN
```

De **GO TO** opdracht in regel 7Ø is erg belangrijk omdat zonder deze opdracht het programma de subroutine tegen zou komen en een foutmelding zou geven (**7 RETURN without GO SUB** = RETURN zonder een GO SUB) zodra de **RETURN** opdracht bereikt wordt.
Hieronder nog een vreemd programma dat het gebruik van **GO SUB** illustreert.

```
100 LET x=10
110 GO SUB 500
120 PRINT s
130 LET x=x+4
140 GO SUB 500
150 PRINT s
160 LET x=x+2
170 GO SUB 500
180 PRINT s
190 STOP
500 LET s=0
510 FOR y=1 TO x
520 LET s=s+y
530 NEXT y
540 RETURN
```

Kijk zelf of u kunt volgen wat dit programma doet, door het te laten lopen. De subroutine begint op regel 5ØØ.
Een subroutine kan zonder problemen een andere subroutine aanroepen, of zichzelf aanroepen (een subroutine die zichzelf aanroept heet *recursief*). U kunt rustig verschillende lagen aanbrengen.

HOOFDSTUK 6

READ, DATA, RESTORE

Samenvatting
READ, DATA, RESTORE

In enkele van de voorgaande programma's hebben we gezien dat informatie, of gegevens (data), direkt aan de computer kunnen worden opgegeven door middel van de **INPUT** opdracht. Soms kan dit erg veel werk zijn, helemaal als een heleboel gegevens elke keer als het programma gerund wordt, herhaald moeten worden. U kunt uzelf veel tijd besparen door gebruik te maken van de **READ, DATA** en **RESTORE** opdrachten. Bijvoorbeeld:

```
10 READ a,b,c
20 PRINT a,b,c
30 DATA 10,20,30
40 STOP
```

Een **READ** (=lees) opdracht bestaat uit een **READ** gevolgd door een lijst met namen van variabelen gescheiden door komma's. Het werkt vergelijkbaar met de **INPUT** opdracht, maar nu hoeft u niet alle waarden die de variabelen moeten krijgen in te tikken. De computer zoekt deze waarden op in de **DATA** (=gegevens) opdracht.
Elke **DATA** opdracht bestaat uit een lijst uitdrukkingen - numerieke of string waarden - die gescheiden zijn door komma's. U kunt zo'n lijst overal in het programma neerzetten, omdat de computer de lijst volledig negeert, totdat hij een **READ** opdracht krijgt die hem zegt de lijst te gaan lezen. Alle gegevens die u achter een of meer **DATA** opdrachten zet vormen samen (ook bij meerdere **DATA** opdrachten) een lange lijst met waarden, de **DATA** *lijst*. De eerste keer dat de computer een **READ** opdracht krijgt, neemt hij de eerste waarde van de **DATA** lijst; de volgende keer neemt hij de tweede waarde. Op deze manier wordt telkens als een **READ** opdracht tegengekomen wordt, een waarde uit de lijst gehaald, zodat de hele **DATA** lijst doorgewerkt wordt. (Als er geprobeerd wordt meer waarden te lezen dan de lijst lang is, volgt een foutmelding.) Merk op dat het verspilde moeite is om **DATA** opdrachten als direkte opdrachten op te geven omdat **READ** de waarden dan niet zal vinden. **DATA** opdrachten moeten in het programma staan.
Laten we eens kijken hoe alles in zijn werk gaat bij het programma dat u net ingetikt heeft. Regel 10 vertelt de computer om drie gegevens te lezen en deze aan de variabelen **a, b** en **c** te geven. Regel 20 geeft dan opdracht om deze variabelen af te drukken (**PRINT**). De **DATA** opdracht in regel 30 geeft de waarden voor **a, b** en **c**. Regel 40 stopt het programma. Om de volgorde waarin dit geheel werkt goed te kunnen

volgen, verandert u regel 2∅ in:

```
20 PRINT b,c,a
```

De informatie in **DATA** kan deel uitmaken van een **FOR..NEXT** lus. Tik:

```
10 FOR n=1 TO 6
20 READ D
30 DATA 2,4,6,8,10,12
40 PRINT D
50 NEXT n
60 STOP
```

Als u een **RUN** geeft kunt u de **READ** opdracht door de **DATA** lijst zien bewegen. **DATA** opdrachten kunnen ook string variabelen bevatten. Bijvoorbeeld:

```
10 READ d$
20 PRINT "De datum is ",d$
30 DATA "1 juni 1982"
40 STOP
```

Dit is een eenvoudige manier om uitdrukkingen van de **DATA** lijst te krijgen: begin bij het begin, werk door de lijst heen tot het einde. Maar het is mogelijk de computer binnen de **DATA** lijst op en neer te laten springen, door middel van de **RESTORE** (=herlaad) opdracht. Deze opdracht bestaat uit **RESTORE** gevolgd door een regelnummer. Door de opdracht gaan de **READ** opdrachten die erop volgen hun gegevens halen vanaf de eerste **DATA** opdracht in of na de regel die opgegeven is in de **RESTORE** opdracht. (Als u het regelnummer weglaat gaat de computer er vanuit dat u de eerste regel van het programma bedoelde.)
Probeer het volgende programma

```
10 READ a,b
20 PRINT a,b
30 RESTORE 10
40 READ x,y,z
50 PRINT x,y,z
60 DATA 1,2,3
70 STOP
```

In dit programma zorgden de gegevens die nodig waren door regel 1∅ ervoor dat **a**=1 en **b**=2. De **RESTORE** 1∅ instruktie zorgt ervoor dat de gegevens weer vanaf het begin van de lijst gelezen worden. **x, y** en **z** krijgen door **READ** hun waarden beginnend bij het eerste getal in de **DATA** opdracht. Laat dit programma nogmaals lopen maar nu zonder regel 3∅ en kijk wat er gebeurt.

HOOFDSTUK 7

Uitdrukkingen

Samenvatting
Bewerkingen: +, -, *, /
Uitdrukkingen, wetenschapelijke notatie, variabele namen

Enkele van de manieren waarop de ZX Spectrum kan rekenen met getallen heeft u al gezien. Hij kan de vier rekenkundige bewerkingen +, -, * en / uitvoeren (onthoud dat * gebruikt wordt voor vermenigvuldigen en / voor delen). Daarnaast kan hij de waarde van een variabele vinden als de naam gegeven is.
Het voorbeeld

```
LET btw=som*19/100
```

geeft slechts een idee van het belangrijke kenmerk dat deze berekeningen gecombineerd kunnen worden. Zo'n combinatie, zoals **som*19/100** wordt een *uitdrukking* (expressie) genoemd. Een uitdrukking is een korte manier om de computer te vertellen verschillende berekeningen uit te voeren, een voor een. In ons voorbeeld betekent **som*19/100** 'zoek de waarde de van de variabele met de naam "som" op, vermenigvuldig dit met 19 en deel het geheel door 100'.
Als u dat nog niet gedaan heeft, raden we u sterk aan eerst het eerste deel van dit boek door te lezen, om te zien hoe de ZX Spectrum getallen behandelt, en in welke volgorde wiskundige uitdrukkingen uitgewerkt worden.
In het kort:
Vermenigvuldigingen en delingen komen eerst. Zij hebben een *hogere rangorde* dan optellingen en aftrekkingen. Relatief ten opzichte van elkaar hebben vermenigvuldiging en deling dezelfde rangorde, hetgeen betekent dat de vermenigvuldigingen en delingen in volgorde van links naar rechts afgewerkt worden. Wanneer deze allemaal gedaan zijn, komen de optellingen en aftrekkingen. Deze hebben onderling ook dezelfde rangorde en gaan dus ook van links naar rechts.
U hoeft alleen te weten of een bepaalde bewerking een hogere of lagere rangorde dan een andere bewerking heeft. De computer doet dit door elke bewerking een getal tussen 1 en 16 te geven en daarmee de rangorde aan te geven. * en / hebben rangorde 8, en + en - hebben rangorde 6.
Deze volgorde is absoluut. U kunt de volgorde echter omzeilen door haakjes te gebruiken: Alles dat tussen haakjes te gebruiken: Alles dat tussen haakjes staat wordt eerst uitgerekend en vervolgens als een enkel getal behandeld.
Uitdrukkingen zijn erg bruikbaar omdat overal waar de computer een getal verwacht van u, ook een bewerking gegeven kan worden. De computer zal deze eerst zelf

uitrekenen. De uitzonderingen hierop zijn zo gering dat deze telkens expliciet aangegeven zullen worden.
U kunt zoveel strings (of stringvariabelen) in een enkele uitdrukking optellen als u zelf wilt. Als u dat wilt kunt u ook haakjes gebruiken.
Het is de hoogste tijd dat we u vertellen wat u wel en wat u niet als naam voor een variabele kunt gebruiken. We hebben al gezegd dat de naam van een stringvariabele moet bestaan uit een enkele letter gevolgd door een dollarteken **$**. De naam van een controle variabele van een **FOR..NEXT** lus moet bestaan uit een enkele letter. De namen van gewone numerieke variabelen zijn veel vrijer. Voor deze namen zijn alle letters en cijfers toegestaan, zolang het eerste teken maar een letter is. U kunt spaties tussenvoegen om de naam leesbaarder te maken, maar de leestekens tellen niet als deel van de naam. Het maakt geen verschil of u de naam in hoofdletters of kleine letters intikt.
Hieronder een paar voorbeelden van toegestane variabele namen.

x

t42

deze naam is zo vreselijk lang dat ik nooit in staat zal zijn om het nog een keer te tikken zonder een vergissing te maken

nu zijn we met zes	(deze laatste twee namen worden als gelijkluidend
nU ZIjn we Met ZeS	beschouwd, en verwijzen naar dezelfde variabele)

De volgende namen zijn niet toegestaan.

2001	(begint met een cijfer)
3 beren	(begint met een cijfer)
M*A*S*H	(* is geen letter of cijfer)
De Vries - Remmers	(- is geen letter of cijfer)

Numerieke uitdrukkingen kunnen weergegeven worden door een getal en een exponent, zie ook deel 1. Probeer het volgende als bewijs:

```
PRINT 2.34e0
PRINT 2.34e1
PRINT 2.34e2
```

en zo verder tot

```
PRINT 2.34e15
```

U zult zien dat ook de computer na verloop van tijd de wetenschappelijke notatie gaat gebruiken. Dit komt omdat er niet meer dan 14 cijfers gebruikt kunnen worden om een getal weer te geven. Probeer ook:

PRINT 2.34e-1
PRINT 2.34e-2

etc.
PRINT geeft slechts 8 significante cijfers van een getal. Probeer

PRINT 4294967295, 4294967295-429e7

Dit bewijst dat de computer de cijfers van 4294967295 wel kan onthouden, maar niet allemaal kan weergeven.
De ZX Spectrum gebruikt vlottende komma rekenkunde, hetgeen wil zeggen dat de cijfers van een getal (de mantisse) en de positie van de decimale komma (de exponent) apart bewaard worden. Dit is niet altijd helemaal exact. Zelfs niet altijd voor gehele getallen. Tik

PRINT 1e10+1-1e10,1e10-1e10+1

Getallen worden opgeslagen tot ongeveer negen en een half cijfer nauwkeurigheid. 1e10 is te groot om exact opgeslagen te worden. De onnauwkeurigheid (in werkelijkheid ongeveer 2) is meer dan 1, dus de getallen 1e10 en 1e10+1 schijnen voor de computer gelijk aan elkaar.
Een nog opvallender voorbeeld is

PRINT 5e9+1-5e9

Hier is de onnauwkeurigheid ongeveer 1. Samen met de 1 die opgeteld wordt, wordt dit *afgerond* tot 2. De getallen 5e9+1 en 5e9+2 zien er voor de computer hetzelfde uit.
De grootste integer (geheel getal) die helemaal accuraat onthouden kan worden is 1 minder dan 32 2-en met elkaar vermenigvuldigd (ofwel 4.294.967.295).
De string "" met helemaal geen tekens erin wordt een *lege* of *nul* string genoemd. Onthoud dat spaties wel degelijk meetellen en dat een lege string dus iets anders is dan een string met alleen spaties.
Probeer

PRINT "Heb je de "Max Havelaar" al uit?"

Als u nu op **ENTER** drukt krijgt u een knipperend vraagteken om aan te geven dat er ergens in de regel een fout gemaakt is. Als de computer de dubbele aanhalingstekens van "Max Havelaar" tegenkomt, denkt hij dat deze het einde van de string "Heb je de" aangeven, en begrijpt hij niet wat 'Max Havelaar' betekent.
Er is een speciale truc om dit probleem op te lossen: Elke keer dat u iets in een string binnen aanhalingstekens wilt zetten, moet u deze twee keer neerzetten, dus:

PRINT "Heb je de ""Max Havelaar"" al uit?"

Zoals u op het beeldscherm kunt controleren worden de extra aanhalingstekens niet weergegeven. U moet hen alleen twee keer tikken om te zorgen dat ze herkend worden.

HOOFDSTUK 8

Strings

Samenvatting
Splitsen, gebruik van **TO** (Deze notatie is geen standaard BASIC !).

Als een bepaalde string gegeven is, dan bestaan de substrings uit een aantal samenstellende karakters van deze string, in volgorde genomen. Dus "string" is een substring van "grotere string", maar "g string" of "grote rode" zijn dat niet.
Er is een notatie *splitsen* (slicing) genoemd om substrings te omschrijven. Deze kan toegepast worden op elke string. De algemene vorm is

string uitdrukking (begin **TO** einde)

(TO = tot). Een voorbeeld hiervan:

"abcdef"(2 TO 5)="bcde"

Als u het getal voor het begin weglaat, wordt het begin van de hoofdstring verondersteld. Als u het getal voor het einde weglaat, wordt het einde van de hoofdstring verondersteld. Dus

"abcdef"(TO 5)="abcdef"(1 TO 5)="abcde"
"abcdef"(2 TO)="abcde"(2 TO 6)="bcdef"
"abcdef"(TO)="abcdef"(1 TO 6)="abcdef"

(Het laatste voorbeeld kunt u ook schrijven als **"abcdef"()**, wat u daar dan ook aan mag hebben).
Een iets andere vorm mist de **TO** en heeft alleen een enkel getal.

"abcdef"(3)="abcdef"(3 TO 3)="c"

Hoewel normaal gesproken zowel het begin als het einde moeten verwijzen naar bestaande delen van de string, zijn er enkele regels die voorrang hebben boven deze normale gang van zaken: Als het begin meer is dan het einde krijgt u een lege string.

"abcdef"(5 TO 7)

geeft een foutmelding **3 subscript wrong** omdat de string slechts 6 karakters heeft, en 7 te veel is, maar bekijk ook

"abcdef"(8 TO 7)=""

en

"abcdef"(1 TO 0)=""

Het begin en het einde mogen niet negatief zijn, anders krijgt u een foutmelding **B integer out of range**.
Het volgende eenvoudige programma laat enkele van deze regels zien.

```
10 LET a$="abcdef"
20 FOR n=1 TO 6
30 PRINT a$(n TO 6)
40 NEXT n
50 STOP
```

Tik **NEW** als dit programma gelopen heeft en voer het volgende programma in:

```
10 LET a$="HET LUKT MIJ"
20 FOR n=1 TO 12
30 PRINT a$(n TO 12),a$((13-n) TO 12)
40 NEXT n
50 STOP
```

Bij string variabelen kunnen we niet alleen substrings uit de hoofdstring halen, maar ook invoegen en veranderen. Tik bijvoorbeeld

LET a$="Ik ben de ZX Spectrum"

en dan

LET a$(7 TO 10)="*****"**

en

PRINT a$

Merk op dat doordat de substring **a$(7 TO 10)** slechts 4 karakters lang is, slechts de eerste vier sterren gebruikt zijn. Dit is een karaktertrek van het toekennen aan substrings: de substring moet exact even lang zijn na het toekennen als hij ervoor was. Om te verzekeren dat dit inderdaad gebeurt, wordt de string waaraan toegekend wordt, aan de rechterkant afgesneden als hij te lang is, of gevuld met spaties als hij te

kort is. Dit wordt de Procrusteaanse toekenning genoemd, naar de herbergier Procrustes die ervoor zorgde dat alle gasten altijd in de bedden pasten door de gasten uit te rekken of hun voeten eraf te hakken.
Als u nu probeert

LET a$()="Hallo daar"

en

PRINT a$;"."

ziet u dat hetzelfde weer gebeurt (deze keer met spaties) omdat **a$()** als substring geldt.

LET a$="Hallo daar"

zorgt ervoor dat het goed gaat.
Ingewikkelde strings hebben soms haakjes nodig voordat ze gesplitst kunnen worden. Bijvoorbeeld:

"abc"+"def"(1 TO 2)="abcde"
("abc"+"def")(1 TO 2)="ab"

Oefening
1. Probeer een programma te schrijven dat telkens de dag van de week afdrukt door gebruik te maken van het string splitsen. Tip: zorg dat de hoofdstring ZonMaaDinWoeDonVryZat is.

HOOFDSTUK 9

Funkties

Samenvatting
DEF
LEN, STR$, VAL, SGN, ABS, INT, SQR
FN

Stel uzelf eens een worstjes machine voor. Aan de ene kant stopt u vlees erin, u draait aan een wiel en aan de andere kant komt er een worstje uit. Een stuk varken zorgt voor een varkensvleesworstje, een stuk vis zorgt voor een visstick en een stuk koe geeft een runderworstje.
Funkties zijn bijna niet te onderscheiden van worstmachines, maar er is een verschil: funkties werken met getallen en strings in plaats van vlees. U stopt er een waarde in (het *argument* genoemd), hakt en mengt het door er enkele berekeningen mee te doen, en er komt een andere waarde uit, het *resultaat*.

Vlees erin --> worstmachine --> worstje eruit

Argument erin --> funktie --> resultaat eruit

Verschillende argumenten geven verschillende resultaten, en als het argument helemaal niet past, stopt de funktie en geeft een foutmelding.
Net als u verschillende machines heeft voor verschillende taken - eentje voor het maken van worstjes, een andere voor het afwassen van de vuile borden, weer een andere voor het bakken van taarten, etc - doen verschillende funkties verschillende berekeningen. Elke heeft zijn eigen waarden om hem te onderscheiden van anderen.
U gebruikt een funktie in een uitdrukking door de naam te tikken gevolgd door het argument. Als de uitdrukking uitgewerkt wordt, wordt het resultaat van de funktie uitgewerkt.
Als voorbeeld een funktie die **LEN** genoemd wordt, die de LENgte van een string berekent. Het argument dat u moet opgeven is de string waarvan u de lengte wilt weten. Het resultaat is de lengte, dus als u tikt

PRINT LEN "Sinclair"

zal de computer als antwoord 8 geven, omdat dat het aantal letters in "Sinclair" is. (Om **LEN** te krijgen moet u, net als met de meeste andere funktienamen, de uitgebreide -extended- modus gebruiken: druk **CAPS SHIFT** en **SYMBOL SHIFT** tegelijk in om de cursor van **L** in **E** te veranderen en druk daarna de **K** toets.)

Als u funkties en bewerkingen in een enkele uitdrukking door elkaar wilt gebruiken, worden de funkties voor de bewerkingen uitgewerkt. Maar ook dit kunt u weer omzeilen door gebruik te maken van haakjes. Hier zijn bijvoorbeeld twee uitdrukkingen die alleen verschillen in de haakjes. Toch worden de berekeningen telkens volledig anders gedaan (de antwoorden zijn toevallig gelijk).

LEN "FRED"+ **LEN** "Bloggs"	**LEN** ("Fred"+"Bloggs")
4+**LEN** "Bloggs"	**LEN** ("FredBloggs")
4+6	**LEN** "FredBloggs"
1Ø	1Ø

Hier nog een paar funkties.

STR$ zet getallen om in strings: Het argument is een getal, en het resultaat is de string die op het scherm zou verschijnen als het getal weergegeven werd door een **PRINT** opdracht. Merk op dat de naam eindigt met een **$** teken om te tonen dat het resultaat een string is. U zou bijvoorbeeld kunnen zeggen

LET a$=STR$ 1e2

hetgeen hetzelfde effect zou hebben als het tikken van

LET a$="1ØØ"

Of u zou kunnen zeggen

PRINT LEN STR$ 1ØØ.ØØØØ

waarbij u het antwoord 3 krijgt omdat **STR$ 1ØØ.ØØØØ="1ØØ"**.

VAL is als **STR$** maar dan omgekeerd: strings worden omgezet in getallen. Bijvoorbeeld:

VAL "3.5"=3.5

In een zeker opzicht is **VAL** het omgekeerde van **STR$** want als u een getal neemt, daar **STR$** op toepast, en daarna **VAL**, krijgt u het originele getal weer terug. Maar als u een string neemt en daar **VAL** op toepast en daarna **STR$** krijgt u niet altijd het origineel als resultaat terug.
VAL is een uiterst krachtige opdracht, omdat de string die het argument voor **VAL** vormt niet per se een getal hoeft te zijn. Numerieke uitdrukkingen zijn ook mogelijk;

VAL "2*3"=6

of zelfs

VAL("2"+"*3")=6

Er zijn hier twee processen aan het werk. Allereerst wordt het argument van **VAL** uitgewerkt als een string: de string uitdrukking **"2"+"*3"** wordt uitgewerkt tot de string **"2*3"**. Vervolgens worden de dubbele aanhalingstekens verwijderd, en het restant uitgewerkt als een getal: **2*3** wordt uitgewerkt tot het getal **6**.
Dit kan knap verwarrend worden als u niet onzettend wakker blijft. Bijvoorbeeld:

PRINT VAL "VAL""VAL""""2"""""""""

(Vergeet niet dat binnen een string een string aanhalingsteken dubbel geschreven moet worden. Als u dieper in de strings gaat zult u merken dat de aanhalingstekens viervoudig of zelfs achtvoudig ingetikt moeten worden.)
Er is nog een funktie die lijkt op **VAL** hoewel die meestal minder nuttig is. Deze funktie heet **VAL$**. Het argument is ook hierbij een string, maar het resultaat is weer een string. Om te kunnen volgen hoe dit werkt moet u nog even voor de geest halen hoe **VAL** in twee stappen gaat: eerst wordt het argument uitgewerkt als een string, dan wordt de string van de aanhalingstekens ontdaan, en wat overblijft wordt uitgewerkt als getal. Met **VAL$** is de eerste stap hetzelfde, maar nadat de string aanhalingstekens verwijderd zijn, wordt het overblijfsel niet als getal maar als een string uitgewerkt. Dus

VAL$"""Vruchtensap""" = "Vruchtensap"

(Merk op hoe belangrijk de aanhalingstekens weer zijn.) Tik

LET a$="99"

en laat de computer de resultaten afdrukken van **VAL a$, VAL "a$", VAL """a$""", VAL$ a$, VAL$ "a$"** en **VAL$ """a$"""**. Enkele hiervan zullen werken, enkele zullen dat niet. Probeer het hoofd koel te houden en leg voor uzelf alle antwoorden uit.
SGN is de *teken* (sign of signum) funktie. Het is de eerste funktie die u ziet die niets met strings te maken heeft. Zowel het argument als het resultaat zijn getallen. het resultaat is +1 als het argument positief is, Ø als het argument nul is en -1 als het argument negatief is.

ABS is een andere funktie waarvan zowel argument als resultaat getallen zijn. De funktie zet het argument om in een positief getal (het resultaat) door het teken te verwaarlozen. Bijvoorbeeld:

ABS -3.2 = ABS 3.2 = 3.2

INT staat voor 'integer deel' - een *integer* is een geheel getal, positief of negatief. Deze

funktie zet een niet geheel getal (het argument) om in een integer (het resultaat) door de breuk te verwaarlozen. Bijvoorbeeld:

INT 3.9 = 3

Wees voorzichtig met deze funktie bij het gebruik van negatieve getallen als argument. De funktie rondt altijd naar beneden af. Dus

INT -3.9 = -4

SQR berekent de *vierkantswortel* (square root) van een getal. Als het resultaat met zichzelf vermenigvuldigd wordt, is de uitkomst het argument. Bijvoorbeeld:

SQR 4 = 2 omdat 2*2=4
SQR Ø.25 = Ø.5 omdat Ø.5*Ø.5=Ø.25
SQR 2 = 1.4142136 (ongeveer)
omdat 1.4142136*1.4142136 = 2.ØØØØØØ1

Als u een willekeurig getal (ook een negatief getal) met zichzelf vermenigvuldigt is het antwoord altijd positief. Dit betekent dat negatieve getallen geen vierkantswortel hebben. Als u een **SQR** probeert toe te passen op een negatief getal krijgt u een foutmelding **An Invalid Argument** (ongeldig argument).

U kunt ook zelf funkties definieren. Als naam kunt u dan kiezen tussen **FN** gevolgd door een letter (als het resultaat een getal is) en **FN** gevolgd door een letter, gevolgd door **$** (als het resultaat een string is). Deze funkties zijn veel strikter in het gebruik van haakjes: het argument moet tussen haakjes staan.
U kunt een funktie definieren door een **DEF** opdracht ergens in het programma te plaatsen. Hieronder als voorbeeld de definitie van een funktie **FN s** waarvan het resultaat het kwadraat van het argument is:

10 DEF FN s(x)=x*x: REM kwadraat van x

DEF krijgt u in de extended modus door **SYMBOL SHIFT** en **1** te gebruiken. Als u dit tikt geeft de computer u automatisch de **FN** omdat in een **DEF** opdracht de **DEF** altijd gevolgd wordt door **FN**. Hierna maakt de **s** de naam **FN s** volledig.
De **x** tussen haakjes is de naam waarmee u naar het argument van deze funktie wilt kunnen verwijzen. U kunt daar een willekeurige enkele letter voor gebruiken (of in het geval van een string argument, een enkele letter gevolgd door **$**).
Na het = teken komt de eigenlijke definitie van de funktie. Dit kan een willekeurige uitdrukking zijn, waarbij tevens naar het argument verwezen worden door de naam te gebruiken die u eraan gegeven heeft (in bovenstaand geval de **x**). Net alsof het een

gewone variabele is.
Als u deze regel ingevoerd heeft, kunt u de funktie in werking stellen net als een van de gewone funkties van de computer door de naam te tikken (**FN s**), gevolgd door het argument. Onthoud dat als u zelf een funktie heeft gedefinieerd, het argument altijd tussen haakjes moet komen te staan. Probeer het een paar keer:

```
PRINT FN s(2)
PRINT FN s(3+4)
PRINT 1+INT FN s(LEN "kuikens"/2+3)
```

Heeft u eenmaal de juiste **DEF** opdracht in het programma gezet, dan kunt u uw eigen funkties net zo veel en gemakkelijk gebruiken als de funkties van de computer.
Merk op: in sommige dialecten van BASIC moeten zelfs de argumenten van de funkties van de computer zelf tussen haakjes gezet worden. Dit is niet het geval bij ZX Spectrum BASIC.
INT rondt altijd naar beneden af. Om af te ronden naar het dichtst bijzijnde gehele getal moet u eerst .5 optellen. U kunt daar uw eigen funktie voor maken.

```
20 DEF FN r(x)=INT(x+0.5):REM x wordt afgerond naar dichtst-
bijzijnde integer.
```

Het resultaat is dan, bijvoorbeeld:

FN r(2.9) = 3	FN r(2.4) = 2
FN r(-2.9) = -3	FN r(-2.4) = -2

Vergelijk deze antwoorden met de antwoorden die u krijgt met **INT** in plaats van **FN r**.
Tik het volgende programma in en laat het lopen

```
10 LET x=0: LET y=0: LET a=10
20 DEF FN p(x,y)=a+x*y
30 DEF FN q()=a+x*y
40 PRINT FN p(2,3),FN q()
```

Er zijn verschillende subtiele punten verwerkt in dit programma.
Allereerst blijkt dat een funktie niet beperkt is tot een argument: een funktie kan meerdere of geen argument hebben. De haakjes moeten echter altijd gebruikt worden.
Ten tweede blijkt dat het er niet toe doet waar in het programma de **DEF** opdracht gezet wordt. Nadat de computer regel 10 gehad heeft, wordt over regel 20 en 30 heen gesprongen om regel 40 uit te voeren. De funktie opdrachten moeten wel ergens in het programma staan. Een funktie kan niet gedefinieerd worden door een direkte opdracht.
Ten derde, **x** en **y** zijn beide namen van variabelen in het hele programma en de na-

men van de argumenten die gebruikt worden voor de funktie **FN p**. In zo'n geval vergeet **FN p** de waarden van **x** en **y** als variabelen tijdelijk. Maar omdat deze funktie geen argument **a** heeft, blijft de waarde van **a** gelijk aan de variabele waarde. Als **FN p(2,3)** uitgewerkt wordt, heeft **a** de waarde 1∅, omdat het een variabele is. heeft **x** de waarde 2 omdat dit het eerste argument is, en **y** heeft de waarde 3 omdat dit het tweede argument is. Het resultaat is dan 1∅ + 2*3 = 16. Als **FN q()** uitgewerkt wordt, zijn er geen argumenten. In dat geval verwijzen **a, x** en **y** nog naar variabelen en hebben dus de waarden 1∅, ∅ en ∅. Het antwoord is dan 1∅ + ∅*∅ = 1∅.
Verander nu regel 2∅ in

```
20 DEF FN p(x,y)=FN q()
```

In dit geval zal **FN p(2,3)** de waarde 1∅ hebben omdat **FN q** terug gaat naar de variabelen **x** en **y** en niet de argumenten van **FN p** gebruikt.
Sommige BASIC dialecten (niet ZX Spectrum BASIC) hebben de funkties LEFT$, RIGHT$, MID$ en TL$.
LEFT$ (a$,n) geeft de substring van a$ die bestaat uit de eerste *n* karakters.
RIGHT$ (a$,n) geeft de substring van a$ die bestaat uit de karakters vanaf het *n*-de karakter.
MID$ (a$,n1,n2) geeft de substring van a$ die bestaat uit *n2* karakters geteld vanaf het *n1*-de karakter.
TL$ (a$) geeft de substring van a$ bestaande uit alle karakters behalve de eerste.
U kunt zelf funkties schrijven om deze taken door de computer uit te laten voeren.
Bijvoorbeeld:

```
10 DEF FN t$(a$)=a$(2 TO ):REM TL$
20 DEF FN l$(a$,n)=a$( TO n):REM LEFT$
```

Controleer of deze funkties werken met strings van een lengte ∅ of 1.
Merk op dat onze **FN l$** twee argumenten heeft, een getal en een string. Een funktie kan maximaal 26 numerieke argumenten hebben (waarom 26?) en tegelijkertijd maximaal 26 string arguments.

Oefening
Gebruik de funktie **FN s(x)=x*x** om **SQR** te testen. U moet vinden

FN s(SQR x)=x

als u een willekeurig positief getal voor **x** invult, en

SQR FN s(x)=ABS x

als **x** een positief of negatief getal is (waarom **ABS**?).

HOOFDSTUK 10

Wiskundige funkties

Samenvatting

↑

PI, EXP, LN, SIN, COS, TAN, ASN, ACS, ATN

Dit hoofdstuk gaat over de wiskundige mogelijkheden van de ZX Spectrum. Het is heel goed mogelijk dat u hiervan nooit iets nodig heeft, dus als u het allemaal iets te ingewikkeld vindt worden, kunt u dit hoofdstuk best overslaan. Dit hoofdstuk behandelt de bewerking ↑ (machtsverheffen), de funkties **EXP** en **LN**, en de goniometrische funkties **SIN, COS, TAN** en hun inversen **ASN, ACS** en **ATN**.

↑ en **EXP**

U kunt een getal verheffen tot de macht van een ander getal - dat betekent 'vermenigvuldig het eerste getal met zichzelf net zo vaak als het tweede getal aangeeft'. Dit wordt normaal gesproken weergegeven door het tweede getal een klein stukje rechtsboven het eerste getal te plaatsen. Zoals u begrijpt is dit wat lastig bij een computer, dus gebruiken we het teken ↑ daarvoor. De machten van twee zijn bijvoorbeeld

2↑1 = 2	
2↑2 = 2*2 = 4	(2 kwadraat, normaal geschreven als 2^2)
2↑3 = 2*2*2 = 8	(2 tot de derde macht, normaal geschreven als 2^3)
2↑4 = 2*2*2*2 = 16	(2 tot de vierde macht, normaal geschreven als 2^4)

Op het meest elementaire niveau betekent '*a*↑*b*', '*a* wordt *b* keer met zichzelf vermenigvuldigd'. Dit is natuurlijk alleen zinvol als *b* een positief geheel getal is. Om een definitie te vinden die ook werkt voor andere waarden van *b*, moeten we de volgende regel bekijken

a↑(b+c) = a↑b*a↑c

(Merk op dat ↑ een hogere rangorde heeft gekregen dan * en /, zodat in een bewerking met meerdere uitdrukkingen, de ↑-en uitgewerkt worden voor de *s en /s.) Het zal niet moeilijk zijn u te overtuigen dat dit werkt als *b* en *c* beiden positieve gehele getallen zijn. Maar als we willen dat dit nog steeds opgaat als ze niet positief of geheel zijn, moeten we nog wat extra's accepteren:

a↑Ø = 1
a↑(-b) = 1/a↑b
a↑(1/b) = de *b*-de wortel van *a*, wat wil zeggen, het getal dat je *b* keer met zichzelf moet vermenigvuldigen om *a* te krijgen.
en
a↑(b*c) = (a↑b)↑c

Als u dit nog nooit eerder gezien heeft, probeer het dan niet in een keer allemaal te onthouden. U hoeft alleen te onthouden:

a↑(-1) = 1/a
en
a↑(1/2) = **SQR** a

Als u deze eenmaal goed door heeft, gaan de anderen na verloop van tijd wel dagen. Experimenteer met dit alles door het volgende programma te proberen:

```
10 INPUT a,b,c
20 PRINT a↑(b+c),a↑b*a↑c
30 GO TO 10
```

Als de regel die we eerder gaven waar is, moeten de twee getallen die de computer elke ronde afdrukt gelijk zijn. (Opmerking - vanwege de manier waarop de computer ↑ uitwerkt, mag het linkergetal - **a** in dit geval - nooit negatief zijn.)
Een typisch gebruiksvoorbeeld van deze funktie is dat van de samengestelde interest. Stel u heeft uw geld in een bouwmaatschappij gestopt, en deze geeft 15% interest per jaar. Na een jaar heeft u niet alleen de 1ØØ% die u toch al had, maar ook de 15% interest die de bouwmaatschappij u gegeven heeft. Samen heeft u dus 115% van het bedrag waarmee u begon. Anders gezegd; U heeft uw geld met een factor 1.15 vermenigvuldigd. Dit is ongeacht de hoeveelheid geld waar u mee begon. Na nog een jaar gebeurt er hetzelfde. U heeft dan 1.15*1.15 = 1.15↑2 = 1.3225 maal uw originele hoeveelheid geld. Na een willekeurig aantal **y** jaren, heeft u **1.15↑y** maal uw startbedrag. Als u de volgende opdracht probeert

```
FOR y=0 TO 100: PRINT y,100*1.15↑y:NEXT y
```

ziet u dat u zelfs met het tamelijk kleine bedrag van Fl.1ØØ,- veel geld kunt verdienen. U merkt ook dat naarmate het bedrag groeit, de stijging ook steeds sneller gaat. (Helaas gaat de inflatie meestal nog sneller.)
Deze ontwikkeling, waarbij na een vaste tijdseenheid, een bepaalde hoeveelheid vermenigvuldigd wordt met een vast deel van die eenheid, wordt *exponentiële groei* genoemd. Dit wordt berekend door een vast getal te verheffen tot een macht aangegeven door de tijdseenheden.
Stel u deed dit:

```
10 DEF FN a(x)=a↑x
```

Hier is *a* min of meer vast. Door middel van **LET** opdrachten kan de waarde, die correspondeert met het interestniveau, af en toe bijgesteld worden.
Er is een bepaalde waarde van **a** die de funktie **FN a** voor een wiskundige extra interessant maakt. Deze waarde wordt *e* genoemd. De ZX Spectrum heeft een funktie **EXP** genaamd die gedefinieerd is als

EXP x = e↑x

Helaas is *e* zelf niet zo'n mooi getal. Het is een oneindige, niet herhalende breuk. De eerste paar decimalen kunt u zien door

PRINT EXP 1

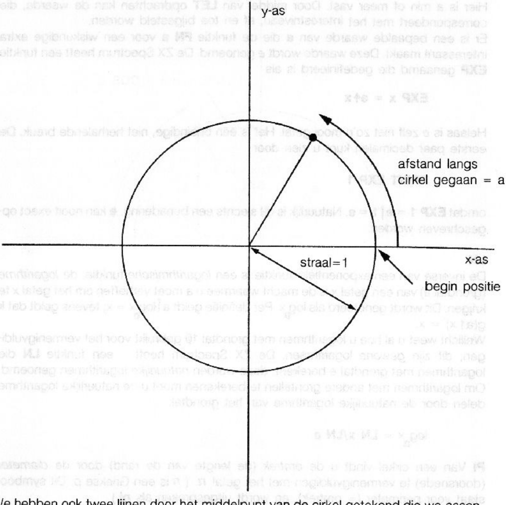
omdat **EXP 1 = e↑1** = **e**. Natuurlijk is dit slechts een benadering. **e** kan nooit exact opgeschreven worden.

LN
De inverse van een exponentiële funktie is een *logarithmische* funktie: de *logarithme* (grondtal *a*) van een getal *x* is de macht waarmee u *a* moet verheffen om het getal *x* te krijgen. Dit wordt genoteerd als $\log_a x$. Per definitie geldt $a↑\log_a x$ = x; tevens geldt dat log(a↑x) = x.
Wellicht weet u al hoe u logarithmen met grondtal 1Ø gebruikt voor het vermenigvuldigen; dit zijn *gewone* logarithmen. De ZX Spectrum heeft een funktie **LN** die logarithmen met grondtal *e* berekent; deze worden *natuurlijke* logarithmen genoemd. Om logarithmen met andere grontallen te berekenen moet u de natuurlijke logarithme delen door de natuurlijke logarithme van het grondtal:

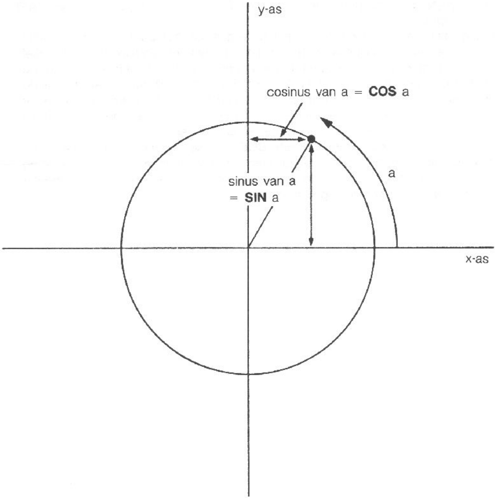
$\log_a x$ = **LN x/LN a**

PI Van een cirkel vindt u de *omtrek* (de lengte van de rand) door de *diameter* (doorsnede) te vermenigvuldigen met het getal π. (π is een Griekse p. Dit symbool staat voor perimeter (= omtrek), en wordt uitgesproken als **pi**.)
Net als *e*, is π een oneindige, niet-herhalende breuk. Het begint als 3.141592653589... Het woord **PI** op de Spectrum (Extended modus, daarna M) staat voor dit getal. Probeer **PRINT PI**.

SIN, COS en **TAN**; **ASN, ACS** en **ATN**
De *goniometrische* funkties berekenen wat er gebeurt als een punt langs een cirkel beweegt. U ziet hier een cirkel met een straal 1 (1 wat? Dat doet er niet toe, zolang dezelfde eenheid maar aangehouden wordt. Niets houdt u tegen om voor elke cirkel die u maakt een nieuwe eenheid te verzinnen), en een punt dat er langs beweegt. Het punt begint op de 3 uur positie, en gaat dan rond tegen de richting van de klok in.



We hebben ook twee lijnen door het middelpunt van de cirkel getekend die we *assen* noemen. De horizontale wordt de *x-as* genoemd en de verticale wordt de *y-as* genoemd.
Om aan te geven waar het punt is, zegt u hoe ver deze langs de cirkel gegaan is, beginnend bij de 3 uur positie (dat is het rechter snijpunt van de x-as met de cirkel). Laten we deze afstand *a* noemen. We weten dat de omtrek van de cirkel 2π is (want de straal is 1, en de doorsnede dus 2), dus als het punt een kwart van de cirkel heeft afgelegd is de afstand $a=\pi/2$. Als het punt de halve cirkel heeft afgelegd is de afstand $a=\pi$, en als het punt de hele cirkel rond geweest is, is de afstand $a=2\pi$.
Als een bepaalde afstand langs de cirkelboog bekend is, zijn nog twee andere afstanden van belang. Hoe ver ligt het punt rechts van de y-as en hoe ver ligt het punt boven de x-as. Deze afstanden worden respectievelijk de *cosinus* en de *sinus* van *a* genoemd. De funkties **COS** en **SIN** berekenen deze afstanden.



Merk op dat als het punt links van de y-as komt, de cosinus negatief wordt, en als het punt beneden de x-as komt, de sinus negatief wordt.
Een ander belangrijk aspect is dat als *a* 2π geworden is, het punt terug is bij de uitgangspositie, en de sinus en cosinus dezelfde waarden weer aannemen:

SIN (a+2*PI) = SIN a
COS (a+2*PI) = COS a

De *tangens* van *a* wordt gedefinieerd als de sinus gedeeld door de cosinus. De funktie die dit uitrekent is **TAN**.
Soms hebben we de inverse van deze funkties nodig. Bij een gegeven sinus, cosinus of tangens moeten we de waarde van *a* vinden. De funkties die dit uitrekenen zijn

arcsinus (**ASN** op de computer), *arccosinus* (**ACS** op de computer) en *arctangens* (**ATN** op de computer).

Kijk nogmaals naar de tekening die het bewegen van het punt *a* langs de cirkel weergeeft, en kijk speciaal naar de lijn die het punt met het middelpunt van de cirkel verbindt. U kunt zien dat de afstand de we *a* genoemd hebben, bepalend is voor de hoek die deze lijn met de x-as maakt. Als $a=\pi/2$ is de hoek 90 graden. Als $a=\pi$ is de hoek 180 graden. En zo de cirkel rond tot $a=2\pi$ en dan is de hoek 360 graden. U kunt de graden eigenlijk wel weglaten, en de hoek uitdrukken in *a*. We zeggen dan dat we de hoek meten in *radialen*. $\pi/2$ radialen is 90 graden, etc.

Vergeet niet dat op de ZX Spectrum **SIN, COS** etc. allemaal met radialen werken en niet met graden. Om van graden naar radialen om te zetten moet u door 180 delen en vermenigvuldigen met π. Om andersom van radialen in graden om te rekenen moet u delen door π en vermenigvuldigen met 180.

HOOFDSTUK 11

Toevalsgetallen

Samenvatting
RANDOMIZE
RND

Dit hoofdstuk gaat over de funktie **RND** en het sleutelwoord **RANDOMIZE**. Beide begrippen worden gebruikt in samenhang met toevalsgetallen. U dient dus op te letten dat u de twee begrippen niet verwart. Beide begrippen bevinden zich op dezelfde toets (**T**); **RANDOMIZE** moest afgekort worden tot **RAND**. In bepaalde opzichten is **RND** net een funktie; het berekent en geeft een resultaat. Meestal heeft **RND** een argument nodig.
Elke keer dat u het gebruikt is het resultaat een toevalsgetal (een willekeurig gekozen getal) tussen Ø en 1. (Soms is het resultaat Ø, maar nooit 1.)
Probeer:

```
10 PRINT RND
20 GO TO 10
```

om te zien hoe de antwoorden kunnen varieren. Kunt u een bepaald patroon ontdekken? Als het goed is, lukt u dat niet; willekeurig gekozen of toeval betekent immers dat er geen patroon is.
In feite is **RND** niet echt willekeurig, omdat het een vaste volgorde van 65536 getallen geeft. Deze zijn echter zo goed door elkaar gegooid, dat er geen patroon te vinden is. We zeggen dan ook dat **RND** *pseudo-toevallig* is.
RND geeft een toevalsgetal tussen Ø en 1, maar u kunt zeer gemakkelijk toevalsgetallen in een ander bereik krijgen. Bijvoorbeeld **5*RND** ligt tussen Ø en 5, en **1.3+0.7*RND** ligt tussen 1.3 en 2. Om gehele getallen te krijgen gebruikt u **INT** (vergeet niet dat **INT** altijd naar beneden afrondt), bijvoorbeeld **1+INT(RND*6)** om te gebruiken in een programma dat een dobbelsteen nabootst. **RND*6** heeft als bereik Ø tot 6, maar omdat 6 zelf nooit bereikt wordt, is **INT (RND*6)** gelijk aan Ø,1,2,3,4 of 5.
Hier is het programma

```
10 REM dobbelsteenprogramma
20 CLS
30 FOR n=1 TO 2
40 PRINT 1+INT(RND*6);" ";
50 NEXT n
60 INPUT a$:GO TO 20
```

Druk op **ENTER** voor elke keer dat u met de 'dobbelsteen' wilt gooien.
De **RANDOMIZE** (=maak toevallig) funktie wordt gebruikt om **RND** op een bepaalde plaats van de serie getallen te laten beginnen, zoals u in onderstaand programma kunt zien.

```
10 RANDOMIZE 1
20 FOR n=1 TO 5: PRINT RND ,: NEXT n
30 PRINT : GO TO 10
```

Na elke keer dat **RANDOMIZE 1** uitgevoerd is, begint de **RND** serie met 0.0022735596. U kunt andere waarden tussen 1 en 65535 met de **RANDOMIZE** opdracht gebruiken om de **RND** serie op andere plaatsen te laten beginnen.
Als u een programma heeft met **RND** erin, en daarin zitten fouten die u niet kunt vinden, dan kunt u de **RANDOMIZE** opdracht op deze wijze gebruiken om ervoor te zorgen dat het programma elke keer als u het laat lopen hetzelfde doet.
RANDOMIZE alleen (hetgeen gelijk is aan **RANDOMIZE 0**) heeft een ander effect, omdat het de **RND** opdracht echt toevallig maakt. U kunt dit in het volgende programma zien.

```
10 RANDOMIZE
20 PRINT RND : GO TO 10
```

De volgorde die u nu krijgt is niet erg willekeurig, omdat **RANDOMIZE** gebruik maakt van de tijd dat de computer aan staat. Omdat elke keer dat de **RANDOMIZE** uitgevoerd wordt, deze min of meer gelijk is, doet de volgende **RND** hetzelfde. U krijgt betere toevalsgetallen als u **GO TO 10** vervangt door **GO TO 20**.
Opmerking: De meeste BASIC dialecten gebruiken **RND** en **RANDOMIZE** om toevalsgetallen te genereren, maar de wijze waarop verschilt wel eens.
Hier een programma dat telkens een munt opwerpt, en het aantal keren kruis of munt telt.

```
10 LET kruis=0: LET munt=0
20 LET geld=INT(RND*2)
30 IF geld=0 THEN LET kruis=kruis+1
40 IF geld=1 THEN LET munt=munt+1
50 PRINT kruis;",";munt,
60 IF munt < >0 THEN PRINT kruis/munt;
70 PRINT : GO TO 20
```

Als u maar lang genoeg doorgaat moet de verhouding tussen kruis en munt 1 naderen. Bij een groot aantal keren proberen is de verwachting immers dat er even vaak kruis als munt boven komt.

Oefeningen
1. Probeer deze regel uit:
Stel u kiest een getal tussen 1 en 872 en u tikt

RANDOMIZE uw getal

De volgende waarde van **RND** is dan

(75*(uw getal+1)-1)/65536

HOOFDSTUK 12

Arrays

Samenvatting
Arrays (de behandeling van stringarrays is iets afwijkend bij de ZX Spectrum).
DIM...

Stel u heeft een lijst met gegevens, bijvoorbeeld de cijfers van tien leerlingen in een klas. Om deze cijfers op te slaan zou u een aparte variabele voor elke leerling kunnen maken, maar dat kan nog wel eens vervelend worden. U zou kunnen proberen om de variabelen Jan 1, Jan 2 etc. tot Jan 1Ø te noemen, maar een programma om deze tien nummers in te tikken zou nogal lang en saai worden.
Het zou zoveel mooier zijn als het volgende programma mogelijk was.

```
 5 REM dit programma zal niet werken
1Ø FOR n=1 TO 1Ø
2Ø READ Jan n
3Ø NEXT n
4Ø DATA 1Ø,2,5,19,16,3,11,1,Ø,6
```

Dit programma is dus niet mogelijk.
Er is echter wel een methode waardoor u zoiets aan kunt pakken, door middel van *arrays* (=orde). Een array is een verzameling variabelen, waarvan de elementen, allemaal met dezelfde naam, alleen van elkaar te onderscheiden zijn door een getal (het *onderschrift*), dat tussen haakjes achter de naam gezet wordt. In ons geval zou de naam *j* kunnen zijn (net als de controle variabelen van **FOR..NEXT** lussen, moet de naam van een array uit een enkele letter bestaan), en de 1Ø variabelen zijn dan *j*(1), *j*(2) tot *j*(1Ø).
De elementen van een array worden variabelen met een *onderschrift* (subscript) genoemd ter onderscheiding van de normale variabelen die u al kent.
Voordat u een array kunt gebruiken, moet u er ruimte voor reserveren in het geheugen van de computer. Dit doet u door middel van de **DIM** (van dimensie) opdracht.

```
DIM b(1Ø)
```

maakt een array met de naam *b* en het formaat (dimensie) 10 (dat wil zeggen dat 10 variabelen met onderschrift zijn; *b*(1),....*b*(10)). De 10 waarden worden gelijk gemaakt aan 0. Elke array met de naam *b* die zich nog in het geheugen bevindt wordt vernietigd. (Een gewone variabele met die naam blijft bestaan. Een array en een gewone numerieke variabele kunnen naast elkaar onder dezelfde naam bestaan. Verwarring is

niet nodig omdat een array variabele altijd een onderschrift heeft.)
Het onderschrift kan gewoon een numerieke uitdrukking zijn. U kunt nu dus tikken

```
 5 DIM j(10)
10 FOR n=1 TO 10
20 READ b(n)
30 NEXT n
40 DATA 10,2,5,19,16,3,11,1,0,6
```

U kunt ook arrays met meer dan een dimensie maken. In een twee dimensionale array heeft u twee getallen nodig om een element aan te kunnen geven - vergelijkbaar met een regel en een kolomnummer van een karakterpositie op het beeldscherm - dus heeft de array de vorm van een tabel. U zou zich ook nog voor kunnen stellen dat de regel- en kolomnummers (twee dimensies) naar een tabel op een bepaalde bladzijde van een boek vol tabellen verwijzen. In dat geval vormen de bladzijdenummers een extra, derde dimensie. We spreken natuurlijk nog steeds over numerieke arrays; de elementen worden niet als karakters in een boek weergegeven, maar zijn getallen. De elementen van een drie dimensionale array *v* worden aangegeven met *v*(bladzijdenummer, regelnummer, kolomnummer).
Voor bijvoorbeeld een twee dimensionale array *c* met de dimensies 3 en 6 gebruikt u de **DIM** opdracht

DIM c(3,6)

Dit geeft u de beschikkiing over 3*6=18 variabelen met onderschrift.

c(1,1),c(1,2),....,c(1,6)
c(2,1),c(2,2),....,c(2,6)
c(3,1),c(3,2),....,c(3,6)

Hetzelfde gaat op voor een willekeurig aantal dimensies.
Hoewel u een nummer en een array met dezelfde naam tegelijk in het geheugen kunt hebben, is het niet mogelijk twee arrays met dezelfde naam te hebben. Ook niet als ze verschillende dimensies hebben.
Er bestaan ook stringarrays. De strings in een array verschillen van gewone strings doordat ze altijd een vaste lengte hebben, en het toekennen hiervan altijd volgens de Procrusteaanse methode gaat - voeten eraf, of opgerekt met spaties. U kunt een stringarray ook zien als een array (met een extra dimensie) met enkele karakters. De naam van een stringarray bestaat uit een enkele letter gevolgd door **$**. In tegenstelling tot de verhouding van numerieke variabelen en numerieke arrays, kunnen een stringvariabele en een stringarray niet dezelfde naam hebben.
Stel u wilt een array **a$** met vijf strings. U moet allereerst beslissen hoe lang de strings moeten zijn. Laten we aannemen dat 10 karakters per string voldoende is. U tikt dan

DIM a$(5,10) (tik dit in)

Dit vormt een 5*10 array met karakters, maar u kunt elke rij ook zien als een string:

a$(1)=a$(1,1) a$(1,2) ... a$(1,10)
a$(2)=a$(2,1) a$(2,2) ... a$(2,10)
. . . .
a$(5)=a$(5,1) a$(5,2) ... a$(5,10)

Als u evenveel onderschriften geeft als er dimensies in de **DIM** opdracht zijn (in dit geval twee), krijgt u een enkel karakter. Maar als u het laatste onderschrift weglaat, krijgt u een string met een vaste lengte. Dus, als voorbeeld, a$(2,7) is het zevende karakter van de string a$(2). Door dit op te schrijven met behulp van de splitsmethode krijgen we a$(2)(7). Tik nu

LET a$(2)="1234567890"
en
PRINT a$(2),a$(2,7)

U krijgt dan

1234567890 7

In plaats van het laatste onderschrift (die u ook helemaal weg kunt laten) kunt u ook een splits-notatie gebruiken, bijvoorbeeld

a$(2,4 TO 8)=a$(2)(4 TO 8)="45678"

Onthoudt:
In een stringarray hebben alle strings dezelfde vaste lengte.
De **DIM** opdracht heeft een extra dimensie (de laatste) om deze lengte aan te geven.
Als u een variabele met onderschrift schrijft voor een string array, kunt u een extra getal, een splitsgetal, gebruiken om te relateren aan het extra getal in de **DIM** opdracht.
U kunt ook stringarrays hebben zonder dimensies. Tik

DIM a$(10)

en u zult zien dat **a$** zich net zo gedraagt als een stringvariabele, met als uitzonderingen dat het altijd een vaste lengte heeft, en de toekenning altijd volgens de methode van Procrustes is.

Oefening
1. Maak gebruik van **READ** en **DATA** opdrachten om een array **m$** met twaalf strings te maken, waarvan **m$(n)** gelijk is aan de **n**de maand van het jaar. (Tip: de **DIM** opdracht zal luiden **DIM m$(12,9)**).

Test dit door alle **m$(n)** af te drukken (gebruik een lus).
Tik

PRINT "In ";m$(5);" leggen alle vogels een ei."

Wat kunt u tegen het teveel aan spaties doen?

HOOFDSTUK 13

Voorwaarden

Samenvatting
AND, OR
NOT

In hoofdstuk 3 zagen we hoe een **IF** opdracht als vorm heeft:

IF voorwaarde **THEN**...

De voorwaarden daarbij waren de vergelijkingen (=, <, >, <=, >=, < >)die twee getallen of strings vergelijken. Verschillende van deze vergelijkingen zijn te combineren met behulp van de logische bewerkingen **AND** (en), **OR** (of) en **NOT** (niet).

Een vergelijking **AND** een andere vergelijking is waar als allebei de vergelijkingen waar zijn. U kunt dus een regel maken als:

IF a$="ja" AND x > Ø THEN PRINT x

Door deze regel wordt **x** alleen afgedrukt als **a$**="ja" en **x** > Ø. BASIC ligt bij deze opdrachten zo dicht bij normaal (Engels) spraakgebruik, dat het de moeite van het uitleggen niet waard lijkt. Net als in de gewone taal kunnen verschillende vergelijkingen aan elkaar gekoppeld worden door middel van **AND**. Het geheel is dan waar als alle individuele vergelijkingen waar zijn.

Een vergelijking **OR** een andere vergelijking is waar als ten minste een van de twee vergelijkingen waar is. (Onthoud dat het totaal ook waar is als allebei de vergelijkingen waar zijn, iets wat in spreektaal niet altijd geldt.)

De **NOT** relatie zet alles op zijn kop. De **NOT** relatie is waar als de vergelijking niet-waar is, en niet-waar als de vergelijking waar is!

Logische uitdrukkingen kunnen gemaakt worden met vergelijkingen en **AND, OR** en **NOT**, net zoals numerieke uitdrukkingen gemaakt kunnen worden met getallen en **+, -** etc. U kunt zelfs gebruik maken van haakjes. Ze hebben een rangorde, net als de gewone bewerkingen +, -, *, / en ↑

OR heeft de laagste rangorde, dan **AND** dan **NOT**, dan de vergelijkingen en dan de gewone bewerkingen.

NOT is een funktie, met een argument en een resultaat, maar de rangorde is veel lager dan die van de andere funkties. Daarom heeft het argument geen haakjes nodig, tenzij in het argument **AND** of **OR** (of allebei) voorkomen. **NOT a=b** betekent hetzelfde als **NOT(a=b)** (en natuurlijk hetzelfde als **a < > b**).

< > is de ontkenning van =; het is waar alleen als = niet-waar is. Met andere woorden,

a < >b is hetzelfde als **NOT a=b**

en tevens

NOT a < >b is hetzelfde als **a=b**

Overtuig uzelf ervan dat **>** = en **<** = de ontkenningen van **>** en **<** zijn. U kunt dus altijd van een **NOT** voor een vergelijking af komen door de vergelijking te veranderen. Ook geldt dat

NOT(een eerste logische uitdrukking **AND** een tweede)

gelijk is aan

NOT(de eerste) **OR NOT**(de tweede)

en

NOT(een eerste logische uitdrukking **OR** een tweede)

is gelijk aan

NOT(de eerste **AND NOT**(de tweede)

Op deze manier kunt u de **NOT**s door de haakjes werken, totdat ze allemaal direkt voor een vergelijking staan, waarna ze weggehaald kunnen worden. Logisch gezien is **NOT** niet strikt noodzakelijk, hoewel een juist gebruik het programma soms duidelijker kan maken.

Het volgende deel is tamelijk ingewikkeld, en kan door leken beter overgeslagen worden. Probeer

PRINT 1=2,1 < >2

waarvan u zou verwachten dat het een foutmelding geeft. In feite bestaat er voor de computer geen logische waarden. In plaats daarvan gebruikt de computer getallen volgens een paar regels.

(i) =, <, >, <=, >= en < > geven allemaal numerieke resultaten: 1 voor waar, en Ø voor niet-waar. De **PRINT** opdracht hierboven drukte dus Ø voor '1=2' af, want dat is niet-waar, en 1 voor '1 < >2' want dat is waar.

(ii) In

IF voorwaarde **THEN**..

kan de voorwaarde elke numerieke uitdrukking zijn. Als de waarde daarvan Ø is, telt het als niet-waar, en elke andere waarde (waaronder de waarde 1 die een ware relatie geeft) telt als waar. De **IF** opdracht hierboven betekent exact hetzelfde als

IF voorwaarde **< >Ø THEN**..

(iii) **AND, OR** en **NOT** zijn ook bewerkingen met getalwaarden.

x **AND** y heeft de waarde
- x, als y waar is (niet nul)
- Ø (niet-waar), als y niet-waar is (nul)

x **OR** y heeft de waarde
- 1 (waar), als y waar is (niet nul)
- x, als y niet-waar is (nul)

NOT x heeft de waarde
- Ø (niet-waar), als x waar is (niet nul)
- 1 (waar), als x niet-waar is (nul)

(Merk op dat 'waar' betekent 'niet nul' als we een gegeven waarde controleren, maar

het betekent '1' als we een nieuwe waarde produceren.)
Lees dit hoofdstuk in het licht van deze uiteenzetting nog eens door, en controleer of alles werkt zoals hier is beschreven.
In de uitdrukkingen **x AND y, x OR y** en **NOT x**, nemen **x** en **y** meestal de waarden 0 en 1 voor niet-waar en waar aan. Werk de tien verschillende combinaties (vier voor **AND**, vier voor **OR** en twee voor **NOT**) uit en controleer of ze doen wat u volgens dit hoofdstuk kunt verwachten dat ze doen.
Probeer het volgende programma

```
10 INPUT a
20 INPUT b
30 PRINT (a AND a > =b)+(b AND a < b)
40 GO TO 10
```

Elke keer wordt de grootste van twee getallen **a** en **b** afgedrukt.
Overtuig uzelf ervan dat u

x AND y

kunt zien als

x if **y** (anders is het resultaat 0)

en

x OR y

kunt zien als

x tenzij **y** (in welk geval het resultaat 1 is)

Een uitdrukking die **AND** of **OR** op deze wijze gebruikt, wordt een *voorwaardelijke* uitdrukking genoemd. Een voorbeeld met **OR** zou er als volgt uit kunnen zien

```
LET totale prijs=prijs zonder BTW*(1.19 OR v$="nul tarief")
```

Merk op hoe **AND** meestal samengaat met optellen (omdat de standaard waarde 0 is), en **OR** meestal samengaat met vermenigvuldigen (omdat de standaard waarde 1 is).
U kunt ook voorwaardelijke uitdrukkingen met stringwaarden maken, maar alleen met **AND**.

x$ AND y heeft de waarde { x$ als y niet nul is
"" als y nul is

Dit betekent **x$** tenzij **y** (anders is de string leeg).
Probeer het volgende programma dat twee strings invoert en hen op alfabetische volgorde zet.

```
10 INPUT "tik twee strings"'a$,b$
20 IF a$ > b$ THEN LET c$=a$: LET a$=b$: LET b$=c$
30 PRINT a$;" ";("<"AND a$ < b$)+("="AND a$=b$);" ";b$
40 GO TO 10
```

Oefening

1. Uit een regel Nederlands of Engels kan BASIC verschillende interpretaties halen. Neem de zin "als a niet gelijk is aan b of c". Hoe zou u deze zin in BASIC kunnen schrijven. Het antwoord is niet

IF A < >B OR C

en ook niet

IF A < >B OR A < >C

HOOFDSTUK 14

De karakterset

Samenvatting
CODE, CHR$
POKE, PEEK
USR
BIN

De letters, cijfers, leestekens etc. die in strings kunnen voorkomen worden *karakters* genoemd. Samen vormen zij het alfabet of de *karakterset* die de ZX Spectrum gebruikt. De meeste van deze karakters zijn enkele *symbolen*, maar er zijn enkele, de *sleutelwoorden* die hele woorden vervangen, zoals **PRINT, STOP, < >** etc.
Er zij 256 karakters en elk karakter heeft een code tussen Ø en 255. Een lijst met alle karakters en de erbij horende codes vindt u in bijlage A. Om van codes naar karakters of andersom over te schakelen zijn er twee funkties, **CODE** en **CHR$**.

CODE wordt gebruikt bij strings, en geeft de code van het eerste karakter van de string (of Ø als de string leeg is).
CHR$ wordt gebruikt bij getallen, en geeft een string (met een enkel karakter) waarvan de code gelijk is aan het gegeven getal.
Het volgende programma drukt de volledige karakterset af.

```
1Ø FOR a=32 TO 255: PRINT CHR$ a;:NEXT a
```

Bovenaan ziet u een spatie, 15 symbolen en leestekens, de 1Ø cijfers, nog zeven symbolen, de hoofdletters, nog zes symbolen, de kleine letters en tenslotte nog 5 symbolen. Deze komen allemaal (behalve **£** en ©) van een veel gebruikte set karakters die bekend staat onder de naam ASCII (dit staat voor American Standard Codes for Information Interchange). ASCII wijst ook numerieke codes voor de karakters toe, en deze codes gebruikt de ZX Spectrum.
De rest van de karakters maken geen deel uit van de ASCII set, en zijn specifiek voor de ZX Spectrum. Allereerst zijn er 15 patronen van zwarte en witte blokken. Deze worden de *grafische* symbolen genoemd omdat ze gebruikt kunnen worden voor het maken van tekeningen. U kunt deze symbolen via het toetsenbord invoeren door gebruik te maken van de zogenaamde *grafische* modus. Als u **GRAPHICS (CAPS SHIFT** met **9**) tikt, verandert de cursor in een **G**. De toetsen voor de 1 tot en met 8 geven nu de grafische symbolen. Alleen een cijfertoets geeft het symbool dat op de toets getekend is. De toets samen met een van de SHIFT toetsen geeft het symbool negatief, dat wil zeggen, wit wordt zwart, zwart wordt wit.

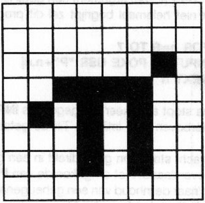
Ongeacht of u een shift gebruikt of niet, de 9 toets zorgt ervoor dat u terug gaat naar de normale (L) modus, en cijfer ∅ is gelijk aan **DELETE**.

Hier ziet u de zestien grafische symbolen.

Symbool	*Code*	*Tik*	*Symbool*	*Code*	*Tik*
	128	G 8		143	G shift 8
	129	G 1		142	G shift 1
	130	G 2		141	G shift 2
	131	G 3		140	G shift 3
	132	G 4		139	G shift 4
	133	G 5		138	G shift 5
	134	G 6		137	G shift 6
	135	G 7		136	G shift 7

Na de grafische symbolen ziet u nog een keer het alfabet van A tot en met U. Dit zijn karakters die u zelf kunt herdefinieren. Als de machine net aangezet is, zijn deze karakters gewoon als letters gedefinieerd. Deze karakters worden *gebruiker-definieerbare* symbolen genoemd. U kunt deze symbolen door het toetsenbord invoeren door in de grafische modus een van de lettertoetsen A tot en met U te gebruiken.
Om een nieuw karakter voor uzelf te definieren volgt u het volgende recept. Dit voorbeeld definieert een karakter als een π.

(i) Bepaal hoe het karakter er uit moet gaan zien. Elk karakter heeft een 8x8 vierkant met punten. Elke punt kan de papierkleur of de inktkleur aangeven (zie deel 1 van dit boek). U kunt een vierkant als hieronder tekenen waarbij de zwarte hokjes de punten aangeven die de inktkleur moeten krijgen.



We hebben een rand van 1 hokje breed aan de randen vrijgelaten omdat alle letters dat zo hebben (behalve de kleine letters met 'staarten', zoals de g, j, p en q, waar de staart helemaal naar beneden doorloopt).
(ii) Bepaal welk karakter de π weer moet gaan geven. Laten we zeggen de P. Als u de P in grafische modus indrukt krijgt u een π..
(iii) Sla het nieuwe patroon op. Elk gebruikers-definieerbaar karakter heeft het patroon opgeslagen als acht getallen, een getal voor elke rij. Elk van deze getallen kan geschreven worden als **BIN** met daarachter acht ∅-en of 1-en, ∅ voor papier en 1 voor inkt. De acht rijen van ons voorbeeld zijn dan

BIN ∅∅∅∅∅∅∅∅
BIN ∅∅∅∅∅∅∅∅
BIN ∅∅∅∅∅∅1∅
BIN ∅∅1111∅∅
BIN ∅1∅1∅1∅∅
BIN ∅∅∅1∅1∅∅
BIN ∅∅∅1∅1∅∅
BIN ∅∅∅∅∅∅∅∅

(Als u bekend bent met binaire getallen, helpt het als u weet dat **BIN** gebruikt wordt om een getal binair in plaats van decimaal weer te geven.)
Deze acht getallen worden in het geheugen opgeslagen, in acht plaatsen. Elke plaats heeft een *adres*. Het adres van de eerste byte, een groep van acht cijfers, is **USR "P"**

(P omdat we dat zo gekozen hebben in (ii)), het adres van de tweede byte is **USR"P"+1**, en zo verder tot het achtste adres dat **USR"P"+7** is.
USR is hier een funktie die ervoor zorgt dat een string argument omzet in het adres van de eerste byte van het corresponderende gebruiker-definieerbaar symbool. Het string argument moet een enkel karakter zijn en kan zowel het door de gebruiker gedefinieerde karakter als de corresponderende letter (kleine letter of hoofdletter) zijn. Er is nog een funktie van **USR** als het argument een getal is. Hier wordt nog op terug gekomen.
Zelfs als u het niet helemaal begrijpt zal dit programma het werk voor u doen:

```
10 FOR n=0 TO 7
20 INPUT rij: POKE USR "P"+n,rij
30 NEXT n
```

Het programma stopt acht keer om gegevens **INPUT** te krijgen, zodat u de acht **BIN** getallen van hierboven kunt intikken. Tik de getallen in volgorde, beginnend met de bovenste rij.
De **POKE** opdracht stopt een getal direkt in een geheugenadres, daarbij de normale BASIC routes overslaand. Het omgekeerde van **POKE** is **PEEK**. Deze opdracht stelt ons in staat om naar de inhoud van een geheugenadres te kijken, zonder dat de inhoud veranderd wordt. Deze beide opdrachten worden uitgebreider behandeld in hoofdstuk 24.
Na de gebruiker-definieerbare karakters komen de sleutelwoorden.
U heeft waarschijnlijk al gemerkt dat we nog niets gezegd hebben over de eerste 32 karakters, met codes van 0 tot en met 31. Dit zijn de *controle* karakters. Zij produceren niet iets dat op het scherm weer te geven is, maar zij bepalen de weergave op het scherm. Soms controleren zij de werking van andere apparatuur dan het beeldscherm, en geeft de televisie een **?** om aan te geven dat het symbool niet begrepen wordt. Voor een volledige beschrijving wordt u verwezen naar bijlage A.
Drie controle karakters die de televisieweergave bepalen zijn belangrijk. Dat zijn de codes 6, 8 en 13. Daarvan is **CHR$ 8** een code die u het meeste nodig zult hebben.
CHR$ 6 drukt spaties af op precies dezelfde wijze als een komma dat doet in een **PRINT** opdracht. Bijvoorbeeld

```
PRINT 1; CHR$6;2
```

geeft hetzelfde resultaat als

```
PRINT 1,2
```

Dit is overduidelijk geen nuttige manier om deze code te gebruiken. Beter is het om te zeggen

```
LET a$="1"+CHR$ 6+"2"
PRINT a$
```

CHR$ 8 is een plaats terug. Door deze opdracht wordt de afdruk positie een plaats terug geschoven. Probeer

PRINT "1234";CHR$ 8;"5"

als resultaat krijgt u **1235**

CHR$ 13 is een nieuwe regel. De afdruk positie wordt verschoven naar het begin van de volgende regel.
De televisie gebruikt hiernaast ook de codes 16 tot en met 23. Meer hierover in de hoofdstukken 15 en 16. Alle controle karakters staan in een lijst in bijlage A.
Door gebruik te maken van de karakter codes kunnen we het idee van alfabetiseren uitbreiden voor strings die niet alleen letters, maar allerlei karakters hebben. Als we in plaats van aan het normale alfabet van 26 letters denken aan het uitgebreide alfabet van 256 karakters in de volgorde van de codes, blijft het principe hetzelfde. De volgende strings staan in ZX Spectrum volgorde. (Merk op dat kleine letter na alle hoofdletters komen. De 'a' komt dus na de 'Z'. En dat spaties uit kunnen maken.)

CHR$ 3+ "ZOOLOGISCHE TUINEN"
CHR$ 8+ "AARDVARKEN JACHT"
" AAAAAAAAAAAHHHHH!"
"(Tussen haakjes)"
"100"
"129.95 inc. BTW"
"AAPJE"
"Aardvarken"
"PRINT"
"Zo zo"
"[tussenvoeging]"
"aardvarken"
"aapje"
"zo zo"
"zoologie"

De regel om uit te zoeken welke string eerst komt is als volgt. Vergelijk eerst de eerste karakters. Als deze verschillen, dan heeft een van de twee een lagere code dan de andere. De string die met een karakter met een lagere code begint, is de lagere string. Als de eerste karakters hetzelfde zijn, ga dan verder met de volgende karakters. Als bij dit proces een van de twee strings op een gegeven moment geen karakters meer heeft, en dus kleiner is dan de andere string, dan is de kortste string de kleinste. Als ook dit niet het geval is, dan moeten de twee strings gelijk aan elkaar zijn .
De vergelijkingen =, <, >, <=, >= en <> worden voor zowel strings als getallen gebruikt: < betekent 'komt eerder' en > betekent 'komt later'. Dus

"AA man" < "AARDVARKEN"
"AARDVARKEN" > "AA man"

zijn allebei waar.

< = en > = werken op dezelfde manier als getallen.

"Hetzelfde" < ="Hetzelfde"

is waar, maar

"Hetzelfde" < "Hetzelfde"

is niet-waar.
Experimenteer met het bovenstaande door het volgende programma in te tikken en te gebruiken. Twee strings worden telkens ingevoerd en op volgorde gezet.

```
10 INPUT "Tik twee strings in:",a$,b$
20 IF a$ > b$ THEN LET c$=a$: LET a$=b$: LET b$=c$
30 PRINT a$;" ";
40 IF a$ < b$ THEN PRINT " < ";: GO TO 60
50 PRINT "=";
60 PRINT " ";b$
70 GO TO 10
```

Merk op dat we in regel 20 **c$** moesten introduceren om **a$** en **b$** te kunnen wisselen.

LET a$=b$: LET b$=a$

zou niet het gewenste effect gegeven hebben.
Het volgende programma maakt gebruiker-definieerbare symbolen om schaakstukken te tonen.

P voor pion
T voor toren
N voor paard
L voor loper
K voor koning
Q voor koningin

Schaakstukken

```
  5 LET b=BIN01111100: LET c=BIN
    00111000: LET d=BIN 00010000
 10 FOR n=1 TO 6: READ p$: REM 6 stukken
 20 FOR f=0 TO 7: REM lees stuk in 8 bytes
 30 READ a: POKE USR p$+f,a
 40 NEXT f
 50 NEXT n
100 REM loper
110 DATA "l",0,d,BIN 00101000, BIN 01000100
120 DATA BIN 01101100,c,b,0
130 REM koning
140 DATA "k",0,d,c,d
150 DATA c, BIN 01000100,c,0
160 REM toren
170 DATA "t",0,BIN 01010100,b,c
180 DATA c,b,b,0
190 REM koningin
200 DATA "q",0, BIN 01010100, BIN 00101000,d
210 DATA BIN 01101100,b,b,0
220 REM pion
230 DATA "p",0,0,d,c
240 DATA c,d,b,0
250 REM paard
260 DATA "n",0,d,c,BIN 01111000
270 DATA BIN 00011000,c,b,0
```

Merk op dat in plaats van **BIN 00000000** ook 0 gebruikt kan worden.
Als u het programma heeft laten lopen, kunt u de stukken bekijken door de grafische modus in te schakelen.

Oefeningen

1. Stel u een ruimte voor een symbool voor dat in vier vierkante delen verdeeld is. Als elk vierkant zwart of wit kan zijn, zijn er 2x2x2x2=16 mogelijkheden om de ruimte te vullen. Zoek deze 16 mogelijkheden in de karakter set op.

2. Run het volgende programma

```
10 INPUT a
20 PRINT CHR$ a;
30 GO TO 10
```

Als u hiermee gaat experimenteren zult u merken dat **CHR$ a** *afgerond* wordt naar het dichtst bijzijnde gehele getal. Als **a** niet binnen het bereik van 0 tot en met 255 valt, stopt het programma met een foutmelding **B integer out of range** (B integer buiten het bereik).

3. Welk van de volgende twee is kleiner?

"ZONDE"
"zonde"

4. Verander het programma zo dat er gebruiker-definieerbare karakters worden gemaakt aan de hand van **READ** en **DATA** opdrachten in plaats van een **INPUT** opdracht.

HOOFDSTUK 15

Meer over PRINT en INPUT

Samenvatting
CLS
PRINT items: helemaal niets
Uitdrukkingen (numerieke of string): **TAB** numerieke uitdrukking, **AT** numerieke uitdrukking, numerieke uitdrukking
PRINT verdelers: **, ; '**
INPUT items: variabelen (numerieke of string)
LINE string variabele
Elk **PRINT** item dat niet met een letter begint. (Sleutelwoorden worden niet gezien als beginnend met een letter.)
SCREEN$

U heeft al flink wat **PRINT** opdrachten gezien, dus u heeft in ieder een geval een idee van de werking. Uitdrukkingen waarvan de waarden worden afgedrukt heten **PRINT** *items*. Zij worden verdeeld door komma's of punt-komma's, die **PRINT** *verdelers* genoemd worden. Een **PRINT** item kan ook uit niets bestaan waardoor hetzelfde gebeurt als u twee komma's achter elkaar zou gebruiken.
Er zijn nog twee soorten **PRINT** items die niet gebruikt worden om de computer te vertellen "wat", maar "waar" hij moet afdrukken. Bijvoorbeeld **PRINT AT 11,16;"*"** zorgt voor het afdrukken van een ster midden op het beeldscherm.

AT regel, kolom

verschuift de **PRINT** positie (de plaats waar het volgende item afgedrukt gaat worden) naar de opgegeven regel en kolom. De regels zijn genummerd van Ø (bovenste regel) tot en met 21, en de kolommen van Ø (links) tot en met 31.

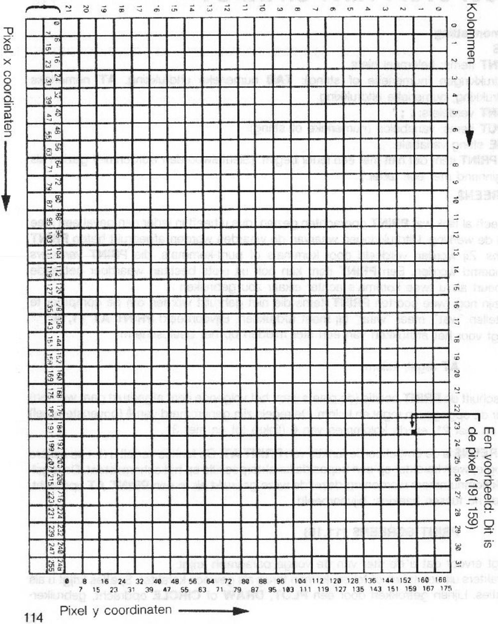
SCREEN$ is de omgekeerde funktie van **PRINT AT**. De funktie vertelt u (tot op zekere hoogte) welk karakter op een bepaalde karakterpositie op het scherm staat. De regel- en kolomnummers worden op dezelfde wijze gebruikt als in een **PRINT AT** opdracht, maar nu tussen haakjes: bijvoorbeeld

PRINT SCREEN$ (11,16)

zorgt ervoor dat u de ster van de vorige paragraph krijgt.
Karakters uit een token of teken, komen terug als gewoon karakter. Spaties krijgt u als spaties. Lijnen getrokken door een **PLOT, DRAW** of **CIRCLE** opdracht, gebruiker-

definieerbare en grafische karakters geven een nul (of lege) string. Als **OVER** gebruikt is om een samengesteld karakter te maken krijgt u ook een lege string.

Normaal gesproken kunt u op de onderste twee regels geen **PRINT** of **INPUT** gebruiken.



TAB kolom

drukt zoveel spaties af dat de **PRINT** positie verschuift naar de opgegeven kolom. De positie blijft op dezelfde regel, tenzij het opgegeven kolomnummer kleiner is dan de huidige positie, dan wordt er naar de volgende regel gesprongen. Merk op dat de computer het opgegeven kolomnummer verkleint met modulus 32 (dat wil zeggen: delen door 32 en de rest nemen); **TAB 33** betekent dus hetzelfde als **TAB 1**. Als voorbeeld

PRINT TAB 3Ø;1;TAB 12;"Inhoud"; AT 3,1;"HOOFDSTUK";TAB 24;"pag."

Dit laat zien hoe u de kop van een inhoudopgave van een boek zou kunnen afdrukken. Probeer dit programma:

```
1Ø FOR n=Ø TO 2Ø
2Ø PRINT TAB 8*n;n;
3Ø NEXT n
```

Dit laat goed zien wat het betekent dat de **TAB** getallen verkleind worden met modulus 32.

Voor een elegantere oplossing kunt u de 8 in regel 2Ø veranderen in een 6.

Enkele kleine punten:

(i) Deze nieuwe items kunnen het beste afgesloten worden met punt-komma's, zoals we hierboven ook gedaan hebben. U kunt ook komma's (of niets, aan het eind van een opdracht) gebruiken, maar dit betekent dat nadat u een **PRINT** positie zorgvuldig heeft vastgelegd, deze onmiddellijk weer laat verschuiven. Dit is meestal niet erg zinvol.

(ii) U kunt niet afdrukken op de onderste twee regels van het scherm (regel 22 en 23) want deze zijn gereserveerd voor opdrachten, **INPUT** gegevens, rapportages etc. Als er hier verwezen wordt naar de onderste regel wordt meestal regel 21 bedoeld.

(iii) U kunt de **AT** gebruiken om de **PRINT** positie te verschuiven naar elk punt, zelfs een punt waar al wat staat. De oude tekst zal gewist worden.

Een andere opdracht die met **PRINT** samenhangt is **CLS**. Deze opdracht maakt het gehele scherm schoon. Dit effect kan ook bereikt worden met **CLEAR** en **RUN**.

Als het afdrukken de onderste regel van het scherm bereikt heeft, begint de tekst naar boven te rollen, net als het papier in een schrijfmachine. U kunt dit goed zien door

CLS: FOR n=1 TO 22: PRINT n: NEXT n

en vervolgens

PRINT 99

een paar keer achter elkaar te gebruiken.

Als de computer enorme hoeveelheden gegevens achter elkaar moet afdrukken, zorgt hij ervoor dat er niets van het scherm af rolt voor u de kans gehad heeft te zien wat er stond. Dit effect ziet u door

CLS: FOR n=1 TO 100: PRINT n: NEXT n

Zodra een scherm vol geschreven is, stopt de computer en vraagt **scroll?** (rollen?) aan de onderkant van het scherm. U kunt nu de eerste 22 getallen op uw gemak bekijken. Als u daarmee klaar bent, tikt u **y** van 'yes' (ja) en de computer geeft u nog een beeldscherm vol getallen. Elke toets die u indrukt zorgt ervoor dat de computer doorgaat, behalve de **n** van 'no' (nee), **STOP** (**SYMBOL SHIFT** en **a**), of **SPACE** (de **BREAK** toets). Deze zorgen ervoor dat de computer stopt met de uitvoering van het programma met een rapportage **D BREAK - CONT repeats**.
De **INPUT** opdracht kan veel meer dan u tot nu toe gezien heeft. U heeft **INPUT** opdrachten gezien in de vorm

INPUT "Hoe oud bent u?",leeftijd

Waardoor de computer de zin **Hoe oud bent u?** onderaan het scherm weergaf, waarna u uw leeftijd kon intikken.
In feite bestaat een **INPUT** opdracht net als een **PRINT** opdracht uit items en verdelers. **Hoe oud bent u?** en **leeftijd** zijn beide **INPUT** items. **INPUT** items zijn normaal gesproken hetzelfde als **PRINT** items maar er zijn enkele belangrijke verschillen.
Ten eerste is een opvallend extra **INPUT** item de variabele waarvan u de waarde moet intikken - in het voorbeeld hierboven **leeftijd**. De regel is dat als een **INPUT** item met een letter begint, het een variabele is waarvan de waarde ingetikt moet worden. Ten tweede; dit zou betekenen dat u de waarden van variabelen niet kunt gebruiken in de opschriften bij een **INPUT** opdracht. U kunt daar onderuit door haakjes rond de variabele te zetten. Elke uitdrukking die met een letter begint moet tussen haakjes gezet worden als deze afgedrukt moet worden als deel van een opschrift.
Elk soort **PRINT** item dat niet door deze regels beinvloedt wordt is tevens een **INPUT** item.
Hier een voorbeeld

LET mijn lft = INT(RND*100):INPUT("Ik ben ";mijn lft;".");"Hoe oud bent u?",uw lft

mijn lft staat tussen haakjes, dus wordt de waarde afgedrukt. **uw lft** staat niet tussen haakjes, dus moet de waarde ingetikt worden.
Alles dat door een **INPUT** opdracht afgedrukt wordt, komt beneden aan het scherm te staan. Dit deel van het scherm werkt min of meer onafhankelijk van het bovenste deel van het scherm. De regels van het onderste deel zijn relatief ten opzichte van de bovenste regel van het bovenste deel van het scherm genummerd, zelfs als deze

bovenste regel ondertussen doorgerold is (hetgeen gebeurt bij een heleboel **INPUT** gegevens).
Om te zien hoe **AT** werkt in een **INPUT** opdracht kunt u het volgende proberen:

```
10 INPUT "dit is regel 1.",a$; AT 0,0;"dit is regel 0.",a$; AT 2,0; "dit is
   regel 2.",a$; AT 1,0;"dit is nog steeds regel 1.",a$
```

(druk op **ENTER** elke keer als er gestopt wordt.) Als **dit is regel 2.** afgedrukt wordt, schuift het onderste deel van het scherm door om ruimte te maken. Maar de benummering schuift ook door, zodat de tekstregels dezelfde nummers houden.
Probeer nu

```
10 FOR n=0 TO 19: PRINT AT n,0;n;: NEXT n
20 INPUT AT 0,0;a$; AT 1,0;a$; AT 2,0;a$; AT 3,0;a$; AT 4,0;a$; AT
   5,0;a$;
```

Terwijl het onderste deel van het scherm hoger en hoger gaat, wordt het bovenste deel niet gestoord, totdat het onderste deel op dezelfde regel dreigt te gaan schrijven als de **PRINT** positie. Dan zal het bovenste deel naar boven rollen om dit te voorkomen.
Een andere verfijning van de **INPUT** opdracht die we tot nu toe nog niet gezien hebben is de **LINE** input. Dit is een andere wijze voor het invoeren van string variabelen. Als u **LINE** voor de naam van een in te voeren string variabele zet, bijvoorbeeld

```
INPUT LINE a$
```

geeft de computer u geen aanhalingstekens die normaal wel gegeven worden. De computer doet voor zichzelf net alsof die aanhalingstekens er wel zijn. Als u dus intikt

kat

als **INPUT** gegeven, zal **a$** de waarde **kat** krijgen. Omdat de aanhalingstekens niet bij de string verschijnen, kunt u hen ook niet weghalen en een andere soort string uitdrukking voor de **INPUT** gegevens intikken. Onthoudt dat u **LINE** niet kunt gebruiken voor numerieke variabelen.
De controle karakters **CHR$ 22** en **CHR$23** hebben effecten vergelijkbaar met **AT** en **TAB**. Als controle karakters zijn ze nogal vreemd, omdat elke keer als er eentje naar de televisie gezonden wordt om afgedrukt te worden, moet hij gevolgd worden door nog eens twee karakters die niet per se hun normale effect hoeven te hebben: Ze worden gezien als getallen (hun codes) om de regel en de kolom aan te geven (bij **TAB**, of de tab positie (bij **TAB**). U zult het bijna altijd eenvoudiger vinden om **AT** en **TAB** op de normale wijze te gebruiken in plaats door middel van hun controle karakters. Er kunnen echter omstandigheden zijn, dat de controle karakters toch van pas komen. Het **AT** controle karakter is **CHR$ 22**. Het eerste karakter erachter geeft het regelnummer weer, en het tweede het kolomnummer. Dus

```
PRINT CHR$ 22+CHR$ 1+ CHR$ c;
```

heeft hetzelfde effect als

```
PRINT AT 1,c;
```

Dit is altijd zo, zelfs als **CHR$ 1** of **CHR$ c** normaal een andere betekenis hebben (bijvoorbeeld als **c=13**); de **CHR$ 22** zorgt ervoor dat op die betekenis niet gelet wordt. Het **TAB** controle karakter is **CHR$ 23** en de twee karakters erachter geven samen een getal tussen Ø en 65535, waarmee het getal aangegeven wordt, dat u anders achter **TAB** gezet zou hebben.

```
PRINT CHR$ 23+CHR$ a+CHR$ b;
```

heeft hetzelfde effect als

```
PRINT TAB a+256*b;
```

U kunt **POKE** gebruiken om de computer te laten stoppen met het telkens om **scroll?** te vragen. U doet dit door

```
POKE 23692,255
```

telkens opnieuw te gebruiken. Hierdoor zal de computer onmiddellijk doorrollen zonder dit eerst te vragen. Probeer als voorbeeld

```
1Ø FOR n=Ø to 1ØØØØ
2Ø PRINT n: POKE 23692,255
3Ø NEXT n
```

en zie hoe alles over uw scherm snelt!

Oefeningen

1. Laat enkele kinderen dit programma proberen, zodat u kunt zien hoe goed zij zijn in vermenigvuldigen.

```
 1Ø LET m$=""
 2Ø LET a=INT(RND*12)+1: LET b=INT(RND*12)+1
 3Ø INPUT (m$)' ' "wat is ";" * ";(b); "?";c
1ØØ IF c=a*b THEN LET m$="Goed.":GO TO 2Ø
11Ø LET m$="Fout. Probeer opnieuw.":GO TO 3Ø
```

Als de kinderen erg oplettend en experimenterend zijn, kunnen ze er achter komen dat ze de oplossingen niet zelf hoeven uit te rekenen. Als de computer hen bijvoorbeeld

vraagt hoeveel 2*3 is, hoeven ze alleen **2*3** in te tikken.
Een van de methoden om dit te voorkomen is om de kinderen strings in plaats van getallen in te laten voeren. Verander **c** in regel 30 door **c$**, en in regel 100 door **VAL c$**. Voeg vervolgens toe

40 IF c$ < >STR$ VAL c$ THEN LET m$="Netjes intikken, als getal!":GO TO 30

Daar zullen ze niet van terug hebben. Maar een echte slimmerik zal na een paar dagen ontdekken dat hier omheen te komen is door de string aanhalingstekens te verwijderen en **STR$ (2*3)** in te tikken. Om ook dit tegen te gaan kunt u **c$** in regel 30 vervangen door **LINE c$**.

HOOFDSTUK 16

Kleuren

Samenvatting
INK, PAPER, FLASH, BRIGHT, INVERSE, OVER
BORDER

Run het volgende programma:

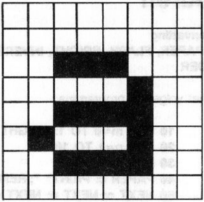
```
10 FOR m=0 TO 1: BRIGHT m
20 FOR n=1 TO 10
30 FOR c=0 TO 7
40 PAPER c: PRINT"    ";:REM 4 gekleurde spaties
50 NEXT c: NEXT n: NEXT m
60 FOR m=0 TO 1: BRIGHT m: PAPER 7
70 FOR c=0 TO 3
80 INK c: PRINT c;"  ";
90 NEXT c: PAPER 0
100 FOR c=4 TO 7
110 INK c: PRINT c;"  ";
120 NEXT c: NEXT m
130 PAPER 7: INK 0: BRIGHT 0
```

Dit programma toont de acht kleuren (inclusief zwart en wit) en de twee helderheden die de ZX Spectrum op een kleurentelevisie kan produceren. (Als uw televisie zwart/wit is, zult u verschillende grijstinten zien.) Hier is een lijst met alle kleuren; de kleuren staan aangegeven op de getal toetsen.

0	-	zwart
1	-	blauw
2	-	rood
3	-	paars
4	-	groen
5	-	licht-blauw, of cyaan
6	-	geel
7	-	wit

Op een zwart/wit televisie komen deze kleuren in volgorde overeen met de mate van helderheid van de grijstint.
Om deze kleuren goed te kunnen gebruiken is wat kennis over de manier waarop een

beeld wordt opgebouwd nodig.
Een beeld is verdeeld in 768 (24 regels van 32) posities waar een karakter geplaatst kan worden, en elk karakter wordt afgedrukt als een 8x8 vierkant van lichtpunten, zoals onderstaande **a**. Dit moet u doen denken aan de gebruiker-definieerbare grafieken van hoofdstuk 14, waarbij een Ø voor een wit puntje stond, en een 1 voor een zwart puntje.



De karakterpositie is geassocieerd met twee kleuren: de *inkt* kleur, ofwel de kleur van de voorgrond hetgeen de kleur van de zwarte puntjes op de tekening is, en de *papier* kleur, ofwel de kleur van de achtergrond, waarmee de witte punten bedoeld worden. Als beginpositie heeft elke positie zwarte inkt en wit papier, zodat het afdrukken verschijnt als zwart op wit.
De karakerpositie heeft ook als gegevens de helderheid (normaal of extra helder) en of er geknipperd moet worden of niet. Knipperen gebeurt door de kleur van inkt en papier snel achter elkaar te wisselen. Deze gegevens worden allemaal in codes gestopt, zodat elke karakterpositie de volgende gegevens kent:
(i) een 8x8 vierkant van Ø-en en 1-en om de vorm van het karakter aan te geven. Ø staat voor papier en 1 staat voor inkt.
(ii) inkt en papier kleuren, elk gecodeerd door een getal tussen Ø en 7.
(iii) helderheid - Ø voor normaal en 1 voor extra helder.
(iv) knippergetal - Ø voor stabiel en 1 voor knipperend.
Merk op dat omdat de inkt en papier kleuren telkens een hele karakterpositie beslaan, het onmogelijk is om meer dan twee kleuren in een bepaald blok van 64 lichtpunten te hebben. Hetzelfde geldt ook voor de helderheid en het al dan niet knipperen: deze verwijzen naar een volledige karakterpositie, niet naar individuele lichtpunten. De kleuren, helderheid en knippergetal van een gegeven positie worden *kenmerken* genoemd.
Als u iets op het scherm afdrukt kunt u het lichtpuntenpatroon op die positie veranderen. Minder opvallend, maar niet minder mogelijk is het om de kenmerken van die positie te veranderen. Als u begint merkt u dat eerst niet omdat alles weergegeven wordt in zwarte inkt op wit papier (normale helderheid en niet knipperend), maar u kunt dit alles varieren met **INK, PAPER, BRIGHT** en **FLASH** opdrachten (inkt, papier,

helderheid en knipper opdrachten). Probeer

PAPER 5

en druk daarna iets af. Alles wat u afdrukt zal op een licht-blauwe achtergrond verschijnen, omdat zodra ze afgedrukt worden de papierkleur van de posities die ingenomen gaan worden op cyaan gezet is (kleurcode 5).
De anderen werken op precies dezelfde manier, dus na

PAPER getal tussen ∅ en 7
INK getal tussen ∅ en 7
BRIGHT ∅ of 1 } zie ∅ als uit
of } en 1 als aan
FLASH ∅ of 1

zal alles wat afgedrukt wordt, het kenmerk bezitten dat opgegeven is, en de positie waar afgedrukt wordt beinvloeden. Probeer verschillende opdrachten uit. U kunt nu zelf het programma aan het begin van het hoofdstuk volgen (bedenk dat een spatie een karakter is met **INK** en **PAPER** in dezelfde kleur).
Er zijn nog een paar getallen die u bij deze opdrachten kunt gebruiken, maar die een iets minder direkt effect hebben.
8 kan bij alle vier de opdrachten gebruikt worden en betekent 'transparant', hetgeen wil zeggen dat het oude kenmerk door het nieuwe kenmerk heenschijnt. Stel dat u tikt

PAPER 8

Geen enkele karakterpositie zal echter de papierkleur op 8 zetten, omdat zo'n kleur niet bestaat. Wat er wel gebeurt is dat als er op die positie afgedrukt wordt, de papierkleur hetzelfde blijft. **INK 8, BRIGHT 8** en **FLASH 8** werken op dezelfde wijze met de andere kenmerken.
9 kan alleen gebruikt worden met **PAPER** en **INK** en betekent 'contrast'. De kleur (inkt of papier) die u hierbij gebruikt wordt contrasterend aan de andere gemaakt bij het afdrukken. Dit gebeurt door de kleur wit te maken als de andere kleur donker is (zwart, blauw, rood of paars) en de kleur zwart te maken als de andere kleur licht is (groen, licht-blauw, geel of wit).
Probeer dit uit door middel van

INK 9: FOR c=∅ TO 7: PAPER c: PRINT c: NEXT c

Nog indrukwekkender kan dit getoond worden door het programma aan het begin van het hoofdstuk te laten lopen om gekleurde strepen te maken en daarna te tikken

INK 9: PAPER 8: PRINT AT ∅,∅;: FOR n=1 TO 1∅∅∅: PRINT n;: NEXT n

De inktkleur wordt hierbij altijd contrasterend aan de papierkleur gemaakt, ongeacht de positie.

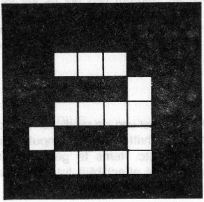
Kleurentelevisies maken gebruik van het feit dat het menselijk oog in feite maar drie kleuren kan zien - de primaire kleuren blauw, rood en groen. De andere kleuren zijn mengkleuren hiervan. Paars wordt bijvoorbeeld gemaakt door blauw en rood te mengen. Daarom is de code ook 3, de som van de codes van blauw en rood.

Om te zien hoe deze kleuren allemaal bij elkaar passen moet u zich drie spotjes voorstellen. Een lamp blauw, een rood en een groen. De lampen beschijnen niet precies dezelfde plaats op een stuk wit papier in een overigens donkere kamer. Daar waar de kleuren elkaar overlappen ziet u mengkleuren. Het volgende programma laat dit zien (merk op dat inkt spaties verkregen worden door een **SHIFT** toets met **8** te gebruiken in de G modus).

```
  10 BORDER 0: PAPER0: INK 7: CLS
  20 FOR a=1 TO 6
  30 PRINT TAB 6; INK 1;"██████████████████": REM 18 inkt
     vierkanten
  40 NEXT a
  50 LET dataline=200
  60 GO SUB 1000
  70 LET dataline=210
  80 GO SUB 1000
  90 STOP
 200 DATA 2,3,7,5,4
 210 DATA 2,2,6,4,4
1000 FOR a=1 TO 5
1010 RESTORE dataline
1020 FOR b=1 TO 5
1030 READ c: PRINT INK c;"██████";:  REM 6 inkt vierkanten
1040 NEXT b: PRINT : NEXT a
1050 RETURN
```

Er is een speciale opdracht **ATTR** waarmee u uit kunt vinden wat de kenmerken (attributes) van een bepaalde karakterpositie zijn. Omdat deze opdracht nogal ingewikkeld is, staat de behandeling ervan achter in dit hoofdstuk.

Er zijn nog twee opdrachten, **INVERSE** en **OVER**, die niet de kenmerken maar het patroon van lichtpuntjes beïnvloeden. Beiden gebruiken 0 voor uit en 1 voor aan, net als **FLASH** en **BRIGHT**. Als u **INVERSE 1** gebruikt zal het lichtpunten patroon negatief afgedrukt worden: papier punten worden inkt punten en andersom. **a** zou dus afgedrukt worden als



Als we zwarte inkt en wit papier (de toestand na het aanzetten) hebben, zal de **a** verschijnen als wit op zwart. Maar deze positie heeft nog steeds zwarte inkt en wit papier. De lichtpunten zijn veranderd.
De opdracht

OVER 1

zorgt voor een speciale manier van afdrukken. Normaal is het zo dat als iets op een karakterpositie gedrukt moet worden waar al iets staat, de oude tekst volledig verwijderd wordt. Na bovenstaande opdracht blijft het oude staan, en wordt het nieuwe er overheen gedrukt (zie echter ook oefening 1). Dit kan zeer handig zijn om bijvoorbeeld samengestelde karakters te maken, zoals letters met een accent erboven. Dit programma drukt een Duitse letter, de o met een umlaut, af. (Tik eerst **NEW**).

```
10 OVER 1
20 FOR n=1 TO 32
30 PRINT "o";CHR$ 8;"""";
40 NEXT n
```

(Let op het controle karakter **CHR 8** om een spatie terug te gaan.)
Er is nog een andere manier om **INK, PAPER** etc. te gebruiken, en deze is voor u misschien nog handiger dan het gebruik als opdracht. U kunt ze namelijk ook als items in een **PRINT** opdracht gebruiken (gevolgd door een **;**). Ze doen dan precies hetzelfde als hierboven uitgelegd is, maar hun effect is slechts tijdelijk, namelijk net zo lang als de **PRINT** opdracht groot is. Als u tikt:

PRINT PAPER 6;"x";:PRINT "y"

zal alleen de x geel afgedrukt worden.
INK en de anderen beinvloeden de kleuren van het onderste deel van het scherm (waar de opdrachten en de **INPUT**s ingetikt worden) niet als ze als opdrachten gebruikt worden. Dit onderste deel van het scherm gebruikt de kleur van de rand (border) voor de papierkleur en code 9 voor contrast inkt. De tekst knippert niet en heeft normale helderheid. De kleur van de rand kan in een van de 8 normale kleuren (niet 8 of 9) veranderd worden door de opdracht

BORDER kleur

Als u **INPUT** gegevens intikt, wordt hierbij de regel van de randkleur voor papier en contrasterende inkt gebruikt. Maar u kunt de opschriften die de computer afdrukt van kleur laten veranderen door **INK** en **PAPER** etc. items te gebruiken in **INPUT** opdrachten, net als bij een **PRINT** opdracht. Het effect blijft tot het einde van de opdracht, of totdat er **INPUT** gegevens ingetikt zijn. Probeer

INPUT FLASH 1; INK 1; "Wat is uw getal?"; n

Er is nog een manier om de kleuren te veranderen door gebruik te maken van controlekarakters - vergelijkbaar met de controlekarakters voor **AT** en **TAB** in hoofdstuk 15.

CHR$ 16 komt overeen met **INK**
CHR$ 17 komt overeen met **PAPER**
CHR$ 18 komt overeen met **FLASH**
CHR$ 19 komt overeen met **BRIGHT**
CHR$ 20 komt overeen met **INVERSE**
CHR$ 21 komt overeen met **OVER**

Elk van deze controlekarakters wordt gevolgd door een karakter dat een kleur aangeeft door middel van een code. Dus (bijvoorbeeld):

PRINT CHR$ 16+CHR$ 9;...

heeft hetzelfde effect als

PRINT INK 9;...

Normaal gesproken hoeft u zich niet druk te maken over deze controlekarakters omdat het gebruik van kleuren in de items veel eenvoudiger is. Maar er is een toepassing waarbij het wel handig is, en dat is om de controlekarakters in een programma te

zetten. het resultaat daarvan is dat verschillende delen van het programma in verschillende kleuren gelist worden. Ze zijn daardoor beter van elkaar te onderscheiden, en het ziet er leuk uit. U moet de codes achter een regelnummer zetten, anders raken ze verloren.
Om de codes in het programma te krijgen moet u ze door middel van het toetsenbord invoeren, meestal door in de uitgebreide modus de cijfers te gebruiken.
De cijfers Ø tot en met 7 veroorzaken de bijhorende kleur - inkt als de **CAPS SHIFT** ingedrukt is, papier als dat niet zo is. Om het iets nauwkeuriger te zeggen: als u in de E modus werkt, en u drukt een cijfer toets in (laten we zeggen de 6 voor geel - alles mag zolang het maar tussen Ø en 7 ligt) worden er twee karakters tussen gevoegd: ten eerste **CHR$ 17** voor **PAPER** en **CHR$ 6** voor het op geel zetten. Als u ook een **CAPS SHIFT** had ingedrukt had u **CHR$ 16** voor 'zet een inkt kleur' gekregen in plaats van **CHR$ 17**.
Omdat het twee karakters zijn, kunt u allerlei vreemde effecten krijgen als u hen weer uitveegt. U moet twee keer **DELETE** indrukken. Vaak krijgt u na de eerste keer indrukken een vraagteken, of zelfs nog vreemdere zaken. Niet op letten; gewoon nog een keer **DELETE** indrukken.
De ← en de → kunnen zich ook raar gedragen als de cursor langs de controlekarakters gaat.
Zolang u nog in de uitgebreide modus bent geldt:

8 geeft **CHR$ 19** en **CHR$ Ø** voor normale helderheid
9 geeft **CHR$ 12** en **CHR$ 1** voor extra helder
CAPS SHIFT met 8 geeft **CHR$ 18** en **CHR$ Ø** voor geen geknipper
CAPS SHIFT met 9 geeft **CHR$ 19** en **CHR$ 1** voor knipperen.

Er zijn ook enkele van deze effecten in de L modus:

CAPS SHIFT met 3 geeft **CHR$ 2Ø** en **CHR$ Ø** voor normale karakters
CAPS SHIFT met 4 geeft **CHR$ 2Ø** en **CHR$ 1** voor inverse karakters.

Samenvattend is hier een complete beschrijving van de bovenste rij van het toetsenbord.

Modus	Shift	1	2	3	4	5	6	7	8	9	0
	SYMBOL	DEF FN	FN	LINE	OPEN #	CLOSE #	MOVE	ERASE	POINT	CAT	FORMAT
E	CAPS	blauwe inkt	rode inkt	paarse inkt	groene inkt	cyaan inkt	gele inkt	witte inkt	geen geknipper	geknipper	zwarte inkt
	GEEN	blauw papier	rood papier	paars papier	groen papier	cyaan papier	geel papier	wit papier	normale helderheid	extra helderheid	zwart papier
G	BEIDE	▙	▟	▄	▛	▌	▞	▖	█	VERLAAT GRAFIEK	DELETE
	GEEN	▝	▘	▀	▗	▐	▚	▜	(leeg)	VERLAAT GRAFIEK	DELETE
K, L of C	CAPS	EDIT	CAPS LOCK	NORMALE VIDEO	INVERSE VIDEO	⇦	⇩	⇧	⇨	GRAFISCHE MODUS	DELETE
	SYMBOL	!	@	#	$	%	&	'	(	)	—
	GEEN	1	2	3	4	5	6	7	8	9	Ø

De **ATTR** funktie heeft als vorm

ATTR (regel kolom)

De twee argumenten zijn de regel en kolom nummers die u zou gebruiken in een **AT** item, en het resultaat is een getal dat de kleur etc. aangeeft van de opgegeven karakterpositie op het televisiescherm. U kunt dit net zo vrij in uitdrukkingen gebruiken als elke andere funktie.

Het getal dat het resultaat vormt is de som van vier andere getallen op de volgende wijze:

128 als de karakter positie knipperend is, ∅ als het stabiel is.

64 als de karakter positie extra helder is, ∅ als het normaal is.

8* de code voor de papier kleur.

1* de code voor de inkt kleur.

Als de karakterpositie bijvoorbeeld knipperend is en normaal met geel papier en blauwe inkt, zijn de vier getallen die opgeteld moeten worden 128, ∅, 8*6=48 en 1, totaal 177. Test dit uit met

PRINT AT ∅,∅; FLASH 1; PAPER 6; INK 1;" ";ATTR (∅,∅)

Oefeningen

1. Probeer

PRINT "B"; CHR$ 8; OVER 1;"/";

Waar de / door de B gegaan is, heeft hij een witte punt open gelaten. Dit is de manier waarop het overdrukken op de ZX Spectrum werkt: twee papieren of twee inkten geven een papier, een van elk geeft een inkt. Dit heeft tevens de interessante eigenschap dat als u iets twee keer met hetzelfde overdrukt, u terug krijgt waar u mee begon. Als u nu tikt

PRINT CHR$ 8; OVER 1;"/"

Krijgt u een onbevlekte **B**. Waarom?

2. Tik

PAPER ∅:INK ∅

- is het niet heel verstandig dat deze effecten de onderste deel van het scherm niet beinvloeden? Tik nu

BORDER ∅

en zie hoe goed de computer op u let!

3. Run het volgende programma

```
10 POKE 22527+RND*704, RND*127
20 GO TO 10
```

Breek uw hoofd niet over de werking van dit programma. De kleuren van de vierkanten op het beeldscherm worden veranderd. De **RND**s zorgen ervoor dat dit willekeurig gebeurt. De diagonale strepen die u af en toe ziet, geven aan dat **RND** niet echt toevallige getallen geeft, maar pseudo-toevallig is.

4. Tik of **LOAD** het schaakstukken programma uit hoofdstuk 14, en tik daarna dit programma in dat een schaakbord weergeeft met de eerder gemaakte schaakstukken erop.

```
  5 REM teken blanko bord
 10 LET bb=1: LET bw=2: REM rood en blauw voor bord
 15 PAPER bw: INK bb: CLS
 20 PLOT 79,128: REM border
 30 DRAW 65,0: DRAW 0,-65
 40 DRAW -65,0: DRAW 0,65
 50 PAPER bb
 60 REM bord
 70 FOR n=0 TO 3: FOR m=0 TO 3
 80 PRINT AT 6+2*n, 11+2*m;" "
 90 PRINT AT 7+2*n, 10+2*m;" "
100 NEXT m: NEXT n
110 PAPER 8
120 LET pw=6: LET pb=5: REM kleuren van witte en zwarte stukken
200 DIM b$(8,8): REM positie van de stukken
205 REM begin posities
210 LET b$(1)="tnlqklnt"
220 LET b$(2)="pppppppp"
230 LET b$(7)="PPPPPPPP"
240 LET b$(8)="TNLQKLNT"
300 REM geef bord weer
310 FOR n=1 TO 8: FOR m=1 TO 8
320 LET bc=CODEb$(n,m): INK pw
325 IF bc=CODE" " THEN GO TO 350: REM spatie
330 IF bc > CODE "Z" THEN INK pb: LET bc=bc-32: REM kleine letters voor
    zwart
340 LET bc=bc+79: REM zet om naar grafiek
350 PRINT AT 5+n,9+m; CHR$ bc
360 NEXT m; NEXT n
400 PAPER 7: INK 0
```

HOOFDSTUK 17

Grafiek

Samenvatting
PLOT, DRAW, CIRCLE
POINT
pixels

In dit hoofdstuk bekijken we hoe we tekeningen met de ZX Spectrum kunnen maken. Het deel van het scherm dat u kunt gebruiken, heeft 22 regels en 32 kolommen, samen zorgend voor 22*32=7Ø4 karakterposities. Uit hoofdstuk 16 weet u dat elke karakterpositie bestaat uit een 8 bij 8 vierkant van lichtpuntjes. Deze lichtpuntjes worden pixels (van picture element' = tekeningelement) genoemd.
Een pixel wordt aangegeven door twee getallen, de *coördinaten*. Het eerste getal, de *x-coördinaat* zegt hoe ver de pixel van de meest linkse kolom af is. De tweede, de *y-coördinaat* zegt hoe ver de pixel van de onderkant (naar boven toe) verwijderd is. Deze coördinaten worden meestal per paar tussen haakjes geschreven. (Ø,Ø), (255,Ø), (Ø,175) en (255,175) zijn respectievelijk de linkeronder-, rechteronder-, linkerboven- en rechterbovenhoek.
De opdracht

PLOT x-coördinaat, y-coördinaat

kleurt de aangegeven pixel. Dit mazelenprogramma tekent een punt elke keer als u **ENTER** indrukt.

```
PLOT INT (RND*256), INT (RND*176): INPUT a$: GO TO 1Ø
```

Het volgende programma is iets interessanter; het tekent een grafiek van de funktie **SIN** (een sinusgolf) voor waarden tussen Ø en 2π.

```
1Ø PLOT n=Ø TO 255
2Ø PLOT n,88+8Ø*SIN (n/128*PI)
3Ø NEXT n
```

Dit programma tekent een grafiek van **SQR** (deel van een parabool) tussen Ø en 4.

```
10 FOR n=0 TO 255
20 PLOT n,80*SQR (n/64)
30 NEXT n
```

Merk op dat de pixel-coördinaten verschillen van de regels en kolommen in een **AT** opdracht. Het rasterpapier in hoofdstuk 15 helpt u uw weg op het scherm te vinden, en de verhouding tussen pixelpositie en karakterpositie te vinden.
Om u te helpen bij het tekenen, kan de computer rechte lijnen, cirkels en delen van cirkels voor u tekenen, waarbij de opdrachten **DRAW** en **CIRCLE** gebruikt moeten worden.
De opdracht **DRAW** (teken) om een rechte lijn te tekenen, heeft als vorm:

```
DRAW x,y
```

Het begin van de lijn is de pixel waar de laatste **PLOT, DRAW** of **CIRCLE** gebleven was (dit wordt de **PLOT** positie genoemd; **RUN, CLEAR, CLS** en **NEW** zetten deze op de beginpositie in de linkerbenedenhoek van het scherm , positie (0,0)). Het eindpunt is **x** pixels naar rechts en **y** pixels naar boven vanaf deze positie. De **DRAW** opdracht bepaalt de richting en de lengte van een lijn, maar niet het beginpunt.
Experimenteer met verschillende **PLOT** en **DRAW** opdrachten, bijvoorbeeld

```
PLOT 0,100: DRAW 80,-35
PLOT 90,150: DRAW 80,-35
```

Opmerking: de getallen in een **DRAW** opdracht kunnen negatief zijn, maar de getallen in een **PLOT** opdracht moeten altijd positief zijn.
U kunt ook in kleur lijnen en punten tekenen, maar u moet daarbij wel onthouden dat een kleur altijd een hele karakterpositie beslaat, en niet op een enkele pixel is in te stellen. Als een pixel getekend wordt, krijgt hij de inktkleur, en wordt de hele karakterpositie waarin deze pixel zich bevindt van dezelfde inktkleur voorzien. Dit programma demonstreert dit effect goed

```
10 BORDER 0: PAPER 0: INK 7: CLS : REM zwart scherm
20 LET x1=0: LET y1=0: REM begin van de lijn
30 LET c=1: REM inkt kleur, om te beginnen blauw
40 LET x2=INT (RND*256): LET y2=INT (RND*176): REM willekeurig
   einde van de lijn
50 DRAW INK c;x2-x1,y2-y1
60 LET x1=x2: LET y1=y2: REM volgende lijn begint bij eindpunt van
   vorige
70 LET c=c+1: IF c=8 THEN LET c=1: REM nieuwe kleur
80 GO TO 40
```

De lijnen lijken steeds breder te worden als het programma langer loopt. Dit komt omdat een lijn alle gekleurde pixels van elke karakterpositie waar de lijn doorgaat beïnvloedt. Merk op dat u **PAPER, INK, FLASH, BRIGHT, INVERSE** en **OVER** in een **PLOT** of **DRAW** opdracht kunt gebruiken, net als bij een **PRINT** of **INPUT** opdracht. U zet ze tussen het sleutelwoord en de coördinaten, en u scheidt hen door middel van puntkomma's of komma's.
Een extra toepassing van **DRAW** is dat u er cirkelbogen (delen van een cirkel) mee kunt tekenen in plaats van rechte lijnen, door een extra getal dat een hoek waarover gedraaid moet worden weergeeft, achter de coördinaten te zetten. De algemene vorm is:

DRAW x,y,a

x en **y** worden gebruikt om het eindpunt van de lijn aan te geven op dezelfde wijze als hiervoor. **a** is het aantal radialen waarover gedraaid moet worden bij het tekenen. Als **a** positief is wordt er linksom gedraaid, als **a** negatief is wordt er rechtsom gedraaid. Een andere manier om tegen **a** aan te kijken, is het te zien als een deel van een complete cirkel. Een hele cirkel is 2π radialen, dus als **a**=π wordt er een halve cirkel getekend, en als **a**=Ø.5*π wordt er een kwart cirkel getekend etc.
Stel als voorbeeld dat **a**=π. Er wordt dan ongeacht de waarden van **x** en **y** een halve cirkel getekend. Run

```
1Ø PLOT 1ØØ,1ØØ: DRAW 5Ø,5Ø,PI
```

waardoor er op uw beeldscherm verschijnt

einde bij (15Ø,15Ø) ,



begin bij (1ØØ,1ØØ)

Het tekenen begint in zuidoostelijke richting, maar tegen de tijd dat het tekenen klaar is, gaat de lijn in noordwestelijke richting: daartussen is de lijn 18Ø graden gedraaid, ofwel π radialen (de waarde van **a**).
Run het programma verschillende keren waarbij **PI** verschillende uitdrukkingen heeft, bijvoorbeeld: **-PI, PI/2, 3*PI/2, PI/4, 1.Ø.**
De laatste opdracht in dit hoofdstuk is de **CIRCLE** (cirkel) opdracht, die een hele cirkel tekent. U geeft de coördinaten van het middelpunt en de straal van de cirkel op in de vorm

CIRCLE x-coördinaat, y-coördinaat, straal

Net als bij **PLOT** en **DRAW** kunt u verschillende kleuritems bij de **CIRCLE** opdracht gebruiken.
De **POINT** funktie vertelt u of een bepaalde pixel de inktkleur of de papierkleur heeft. De funktie kent twee argumenten; de coördinaten van de te onderzoeken pixel (welke tussen haakjes moeten staan). Het resultaat is Ø als de pixel de kleur van het papier heeft, en 1 als de pixel inktkleur heeft. Probeer

CLS : PRINT POINT (Ø,Ø): PLOT Ø,Ø : PRINT POINT (Ø,Ø)

Tik

PAPER 7: INK Ø

en laten we eens kijken hoe de **INVERSE** en de **OVER** werken binnen een **PLOT** opdracht. Deze twee opdrachten beïnvloeden alleen de aangegeven pixel en niet de rest van de karakterpositie. Normaal gesproken zijn ze uit (Ø) in een **PLOT** opdracht, dus u hoeft ze alleen maar aan te geven om ze aan te zetten (1).
Hier een lijstje met mogelijkheden om na te kunnen slaan:
PLOT; - dit is de gewone vorm. Een inkt lichtpunt wordt getekend, d.w.z. dat de pixel in de inktkleur weergegeven wordt.
PLOT INVERSE 1; - dit tekent een puntje uitwisvloeistof, d.w.z. de pixel krijgt de kleur van het papier.
PLOT OVER 1; - dit verwisselt de kleur van de pixel: als het papierkleur was, wordt het inktkleur en andersom.
PLOT INVERSE 1; **OVER** 1; - dit laat de pixel precies als hij was, maar het zorgt wel voor een verandering van de **PLOT** positie.

Voor nog een voorbeeld van het gebruik van de **OVER** opdracht moet u het scherm vullen met zwarte tekst op witte achtergrond. Hierna tikt u

PLOT Ø,Ø: DRAW OVER 1;255,175

Hiermee krijgt u een redelijk nette lijn. Alleen daar waar de lijn de geschreven tekst raakt, vallen gaten. Tik dezelfde opdrachten precies zo nog een keer in. De lijn verdwijnt zonder een spoor achter te laten. Dit is het grote voordeel van **OVER 1**. Als u de lijn getekend had door middel van

PLOT Ø,Ø: DRAW 255,175

en de lijn gewist had met

PLOT Ø,Ø: DRAW INVERSE 1;255,175

dan had u ook delen van de tekst gewist.

Probeer nu

PLOT ∅,∅: DRAW OVER 1;25∅,175

en probeer de lijn weer uit te wissen met

DRAW OVER 1;-25∅,-175

Dit werkt niet echt goed omdat de pixels die de lijn op de terugweg gebruikt niet exact hetzelfde zijn als de gebruikte pixels op de heenweg. U moet een lijn in precies dezelfde richting zowel tekenen als wissen.
Een manier om meer dan de normale 8 kleuren te krijgen is een mengkleur te maken door in een vierkant spikkeltjes te maken, met een gebruiker-definieerbaar karakter. Voer het volgende programma uit

```
1∅∅∅ FOR n=∅ TO 6 STEP 2
1∅1∅ POKE USR "a"+n,BIN ∅1∅1∅1∅1: POKE USR "a"+n+1, BIN
     1∅1∅1∅1∅
1∅2∅ NEXT n
```

Dit geeft een gebruiker-definieerbaar karakter dat eruit ziet als een schaakbord(je). Als u dit karakter (grafische modus, dan **a**) in rode inkt op geel papier afdrukt, zult u zien dat het een redelijke mengkleur oranje geeft.

Oefeningen
1. Experimenteer met **PAPER, INK, FLASH** en **BRIGHT** items in een **PLOT** opdracht. Deze onderdelen beïnvloeden de hele karakterpositie waarin de pixel zich bevindt. Normaal gesproken lijkt het of de **PLOT** opdracht begonnen is met

PLOT PAPER 8; FLASH 8; BRIGHT 8;...

en dat alleen de inktkleur van een karakterpositie verandert als er iets getekend wordt. Maar u kunt dit naar willekeur veranderen.
Wees extra voorzichtig als u kleuren gebruikt met **INVERSE 1**, want dit zorgt ervoor dat de pixel de papierkleur geeft, maar verandert de inktkleur, en dit is niet altijd wat u zou verwachten.

2. Probeer cirkels te tekenen met behulp van **SIN** en **COS** (probeer zelf uit te zoeken hoe, na het lezen van hoofdstuk 1∅). Run dit programma

```
1∅ FOR n=∅ TO 2*PI STEP PI/18∅
2∅ PLOT 1∅∅+8∅*COS n,87+8∅*SIN n
3∅ NEXT n
4∅ CIRCLE 15∅,87,8∅
```

U ziet dat de **CIRCLE** opdracht misschien wel iets minder precies is, maar stukken sneller.

3. Probeer

CIRCLE 100,87,80: DRAW 50,50

Hiermee kunt u zien dat de **CIRCLE** opdracht de **PLOT** positie op een tamelijk onduidelijke plaats laat - altijd ergens ongeveer halverwege de rechterkant van de cirkel. Meestal heeft u een **PLOT** opdracht na een **CIRCLE** opdracht nodig om verder te kunnen tekenen.

4. Hier volgt een programma dat de grafiek van bijna elke funktie kan tekenen. Het programma vraagt u eerst om een getal **n**; de funktie zal getekend worden van **-n** tot **+n**. Daarna vraagt het programma de funktie zelf in te voeren als een string. De string moet een uitdrukking zijn met **x** als argument van de funktie.

```
10 PLOT 0,87: DRAW 255,0
20 PLOT 127,0: DRAW 0,175
30 INPUT s,e$
35 LET t=0
40 FOR f=0 TO 255
50 LET x=(f-128)*s/128: LET y=VAL e$
60 IF ABS y>87 THEN LET t=0: GOTO 100
70 IF NOT t THEN PLOT f,y+88: LET t=1: GO TO 100
80 DRAW 1,y-old y
100 LET old y=INT (y+.5)
110 NEXT f
```

Run dit programma en tik als voorbeeld **10** in voor het getal **n** en **10*TAN x** als funktie. Het programma zal de grafiek van tan *x* tekenen voor $-10 < x < +10$.

HOOFDSTUK 18

Beweging

Samenvatting
PAUSE, INKEY$, PEEK

U zult vaak het programma een tijdje willen laten wachten. Daarbij komt de **PAUSE** opdracht (pauze) van pas.

PAUSE n

stopt het computeren en geeft het dan aanwezige beeld *n* keer weer (5Ø beelden per seconde in Europa en 6Ø beelden per seconde in Amerika). *n* kan maximaal 65535 zijn, hetgeen zorgt voor een pauze van bijna 22 minuten. Als *n*=Ø, betekent dit 'eeuwige **PAUSE**'.
Een pauze kan altijd onderbroken worden door een toets in te drukken (let op dat een **CAPS SHIFT** met de spatiebalk tevens voor een break zorgt). U moet de toets indrukken nadat de pauze is begonnen.
Dit programma laat de secondenwijzer van een klok rondlopen:

```
1Ø REM eerst tekenen we de klok
2Ø FOR n=1 TO 12
3Ø PRINT AT 1Ø-1Ø*COS (n/6*PI),16+1Ø*SIN (n/6*PI);n
4Ø NEXT n
5Ø REM nu starten we de klok
6Ø FOR t=Ø TO 2ØØØØØ: REM t is de tijd in seconden
7Ø LET h=t/3Ø*PI : REM h is de hoek van de secondenwijzer in
   radialen
8Ø LET sx=8Ø*SIN h: LET sy=8Ø*COS h
2ØØ PLOT 128,88: DRAW OVER 1;sx,sy: REM teken wijzer
21Ø PAUSE 42
22Ø PLOT 128,88: DRAW OVER 1;sx,sy: REM wis wijzer
4ØØ NEXT t
```

Deze klok houdt er na ongeveer 55,5 uur mee op door regel 6Ø, maar u kunt hem gemakkelijk langer laten lopen. Merk op hoe de timing gecontroleerd wordt door regel 21Ø. U zou wellicht **PAUSE 5Ø** verwachten om een seconde aan te geven, maar het be-

rekenen kost ook een beetje tijd, en dat moet ingecalculeerd worden. Dit kunt u het beste doen door te proberen; de computerklok telkens vergelijken met een echte klok, en regel 210 telkens aanpassen totdat ze gelijk lopen. (Dit kan niet erg precies; een aanpassing van een beeldje in een seconde is gelijk aan 2% of een half uur per dag.) Er is een andere veel nauwkeurigere manier om de tijd te meten. Deze maakt gebruik van de inhoud van bepaalde geheugenlokaties. De opgeslagen gegevens worden opgehaald door middel van een **PEEK**. Hoofdstuk 25 legt uitgebreid uit, wat we hier doen. De gebruikte uitdrukking is:

(65536*PEEK 23674+256*PEEK 23673+PEEK 23672)/50

Dit geeft het aantal seconden sinds de computer aangezet werd (tot ongeveer 3 dagen en 21 uren, waarna de waarde weer terug gaat naar 0).
Hier een verbeterd klokprogramma om hier gebruik van te maken:

```
10 REM eerst de klok tekenen
20 FOR n=1 TO 12
30 PRINT AT 10-10*COS (n/6*PI),16+10*SIN (n/6*PI);n
40 NEXT n
50 DEF FN t()=INT ((65536*PEEK 23674 + 256*PEEK 23673 + PEEK
   23672) /50) : REM aantal seconden vanaf het begin
100 REM nu starten we de klok
110 LET t1=FN t()
120 LET h=t1/30*PI : REM h is hoek van secondenwijzer
130 LET sx=72*SIN h: LET sy=72*COS h
140 PLOT 131,91: DRAW OVER 1;sx,sy: REM teken wijzer
200 LET t=FN t()
210 IF t<=t1 THEN GO TO 200: REM wacht tot het tijd is voor de
    volgende wijzer
220 PLOT 131,91: DRAW OVER 1;sx,sy: REM wis oude wijzer
230 LET t1=t : GO TO 120
```

De interne klok die bij deze methode gebruikt wordt, is accuraat tot ongeveer .01% zolang de computer een programma aan het uitvoeren is. Dit wil zeggen 10 seconden per dag. Maar de klok stopt tijdelijk elke keer dat u een **BEEP** doet, of een cassettebewerking, of de printer gebruikt, of enige andere randapparatuur. Al deze handelingen zorgen ervoor dat de klok achter gaat lopen.
De getallen **PEEK 23674, PEEK 23673** en **PEEK 23672** worden in de computer bewaard en gebruikt om in 50-sten van seconden te tellen. Elk kan van 0 tot 255 tellen, en ze worden telkens eentje verhoogd totdat 255 bereikt is, waarna ze weer op 0 springen.
Degene die het vaakste verspringt is **PEEK 23672**. Elke 1/50 seconde wordt deze met 1 verhoogd. Zodra 255 bereikt is, zorgt de volgende verhoging dat weer naar 0 gesprongen wordt en tegelijkertijd **PEEK 23673** met eentje verhoogd wordt. Elke keer

(elke 256/50 seconden) dat **PEEK 23673** van 255 naar 0 springt, wordt **PEEK 23674** met eentje verhoogd. Hiermee is duidelijk waarom en hoe de uitdrukking hiervoor werkt.
Overdenk het volgende zorgvuldig: Stel, onze drie getallen zijn 0 (voor **PEEK 23674**), 255 (voor **PEEK 23673**) en 255 (voor **PEEK 23672**). Dit betekent dat het ongeveer 21 minuten na het aanzetten is. Onze uitdrukking zou als resultaat moeten hebben

(65536*0+256*255+255)/50=1310,7

Maar er is een verborgen gevaar. De volgende keer dat er een 1/50 telling is, zullen de drie getallen veranderen in 1, 0 en 0. Dit zou wel eens precies halverwege het evalueren van de uitdrukking kunnen zijn: de computer zou **PEEK 23674** uitwerken als 0, maar daarna de twee anderen veranderen in 0 voordat naar de geheugenadressen gekeken (PEEK) kan worden. Het antwoord zou dan zijn

(65536*0+256*0+0)/50=0

hetgeen natuurlijk vreselijk fout is.
Een eenvoudige regel om dit probleem te voorkomen is: *werk de uitdrukkig twee keer uit, en neem het grootste resultaat.*
Als u precies naar het programma hiervoor kijkt, ziet u dat het programma dit impliciet doet.
Hier een truc om de regel toe te passen. Definieer de funkties als volgt

```
10 DEF FN m(x,y)=(x+y+ABS (x-y))/2: REM de grootste van x en y
20 DEF FN u() =(65536*PEEK 23674+256*PEEK 23673+PEEK
   23672)/50: REM de tijd, die verkeerd kan zijn
30 DEF FN t()=FN m(FN u(), FN u()): REM de tijd, goed
```

U kunt de drie tellers veranderen zodat ze de werkelijke tijd in plaats van de tijd sinds het aanzetten van de computer aangeven. Om als tijd bijvoorbeeld 10.00 uur 's ochtends in te stellen, berekent u eerst dat dit 10*60*60*50=180000
050-sten van seconden is, en dat

1800000=65536*27+256*119+64

Om de drie tellers op 27, 119 en 64 in te stellen, tikt u

POKE 23674,27: POKE 23673,119: POKE 23672,64

In landen waar de netspanning een frequentie van 60 Hertz heeft, moet u in de programma '50' door '60' vervangen.
De funktie **INKEY$** (die geen argument heeft) leest iets van het toetsenbord. Als u precies een toets intikt (of een **SHIFT** met een andere toets) is het resultaat het

karakter dat die toets in de L-modus geeft, anders is het resultaat een lege of nulstring. Probeer dit programma dat als een schrijfmachine werkt

```
10 IF INKEY$ <>"" THEN GO TO 10
20 IF INKEY$ = "" THEN GO TO 20
30 PRINT INKEY$ ;
40 GO TO 10
```

In dit programma wacht regel 10 tot u uw vinger van de toets afhaalt, en regel 20 wacht tot u een nieuwe toets indrukt.
Onthoud dat in tegenstelling tot **INPUT**, **INKEY$** niet op u wacht. U hoeft dus geen **ENTER** te tikken, maar als u niets intikt, heeft u uw kans laten lopen.

Oefeningen

1. Wat gebeurt er als u regel 10 weglaat in het schrijfmachine programma?

2. U kunt de **INKEY$** funktie ook gebruiken samen met de **PAUSE**, zoals in dit alternatieve schrijfmachineprogramma.

```
10 PAUSE 0
20 PRINT INKEY$
30 GO TO 10
```

Waarom is het belangrijk dat de pauze niet ophoudt als u al een toets ingedrukt heeft wanneer het programma begint?

3. Verander het secondenwijzerprogramma zodat het ook de minuten- en urenwijzer aangeeft. Als u er wat voor voelt, kunt u het programma zo aanpassen dat elke kwartier iets bijzonders gebeurt. Met behulp van de **BEEP** opdracht (zie volgende hoofdstuk) kunt u de bekende tonen van de Big Ben laten klinken.

4. (voor sadisten) Probeer dit:

```
10 IF INKEY$ ="" THEN GO TO 10
20 PRINT AT 11,14; "OOOOOHHH!!"
30 IF INKEY$ <> "" THEN GO TO 30
40 PRINT AT 11,14;" "
50 GO TO 10
```

HOOFDSTUK 19

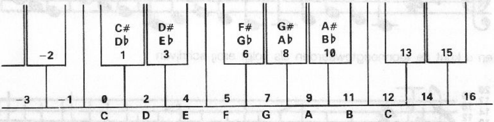
BEEP

Samenvatting
BEEP

Als u nog niet had ontdekt dat de ZX Spectrum een ingebouwde luidspreker heeft, moet u eerst deel 1 van dit boek doorlezen voor u verder gaat.
De luidspreker geeft geluid door de **BEEP** (piep) opdracht te gebruiken.

BEEP duur, hoogte

waarbij, zoals gebruikelijk, 'duur' en 'hoogte' een numerieke uitdrukking voorstellen. De duur slaat op de tijdsduur en wordt in seconden uitgedrukt. De hoogte slaat op de hoogte van de toon ('pitch') en wordt in halve tonen vanaf de midden C gegeven.
Hier vindt u een tekening om de verschillende hoogtes van alle noten van een octaaf op de piano weer te geven.
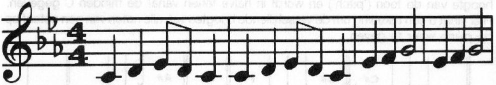


Om hogere of lagere tonen te krijgen, moet u 12 optellen of aftrekken voor iedere octaaf die u hoger of lager wilt hebben.
Als u een piano in de buurt heeft als u aan het programmeren bent, is deze tekening waarschijnlijk voldoende om alle toonhoogten te kunnen programmeren. Als u direkt van bladmuziek wilt schrijven, is het misschien beter dat u een tekening maakt van een notenbalk met daarop aangegeven de verschillende toonhoogten, en de bijhorende waarden.
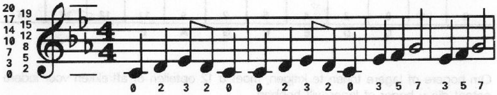
Tik bijvoorbeeld

```
10 PRINT " Vader Jacob"
20 BEEP 1,0: BEEP 1,2: BEEP 5,3: BEEP 5,2: BEEP 1,0
30 BEEP 1,0: BEEP 1,2: BEEP 5,3: BEEP 5,2: BEEP 1,0
40 BEEP 1,3: BEEP 1,5: BEEP 2,7
50 BEEP 1,3: BEEP 1,5: BEEP 2,7
60 BEEP .75,7: BEEP .25,8: BEEP .5,7: BEEP .5,5: BEEP .5,3:
   BEEP .5,2: BEEP 1,0
70 BEEP .75,7: BEEP .25,8: BEEP .5,7: BEEP .5,5: BEEP .5,3:
   BEEP .5,2: BEEP 1,0
80 BEEP 1,0: BEEP 1,-5: BEEP 2,0
90 BEEP 1,0: BEEP 1,-5: BEEP 2,0
```

Als u dit programma runt, moet u de dodenmars van Mahlers eerste symfonie krijgen, het deel waarin de kabouters de US-soldaten begraven.
Stel dat uw wijsje in C mineur geschreven is, zoals het stuk van Mahler hiervoor. Het begin ziet er zo uit:



en u kunt de toonhoogtewaarden als volgt erbij schrijven



We hebben twee hulplijnen getekend om de waarden goed aan te kunnen geven. Let op hoe de E mol in de zetting niet alleen de E bovenaan beinvloedt, door hem van 16 naar 15 te halen, maar ook de E op de onderste regel, deze van 4 naar 3 halend. Op deze wijze moet het niet moeilijk zijn om de waarde van elke noot te vinden.
Als u de zetting van een stuk wilt veranderen, is het eenvoudiger om een variabele **key** (sleutel) te nemen en **key+** voor elke toonwaarde te zetten. De tweede regel wordt dan

```
20 BEEP 1,key+0: BEEP 1,key+2: BEEP .5,key+3: BEEP .5,key+2:
   BEEP 1,key+0
```

Voordat u het programma zo kunt laten uitvoeren, moet u **key** eerst een waarde geven. Ø voor C mineur, 2 voor D mineur, 12 voor C mineur een octaaf hoger, etc. U kunt de computer op deze wijze ook stemmen met een ander instrument door **key** met hele kleine stappen aan te passen.
U moet ook de tijdsduur van elke toon uitzoeken. Omdat dit stuk tamelijk langzaam is, hebben we een seconde voor een kwartnoot genomen, en de rest daarop gebaseerd; een halve seconde voor een achtste noot, etc.

Het is flexibeler om een variabele **kwart** te nemen om daar de lengte van een kwartnoot in op te kunnen slaan, en de tijdsduur hierin uit te drukken. Regel 2Ø zou dan worden

```
2Ø BEEP kwart, key+Ø: BEEP kwart,key+2 : BEEP kwart/2, key+3 :
   BEEP kwart/2, key+2 : BEEP kwart, key+Ø
```

(Het is natuurlijk nog sneller om **kwart** en **key** kortere namen te geven).
Door **kwart** de juiste waarden te geven, kunt u de snelheid van het stuk goed regelen.
U moet onthouden dat er maar een luidspreker in de computer zit, zodat u maar een noot tegelijk kunt laten klinken. U bent gebonden aan niet-harmonischen. Als u er meer wilt, zult u moeten zingen.
Probeer zelf wat wijsjes te programmeren. Begin bij iets eenvoudigs als 'boer er ligt een kip in het water'. Als u geen piano en ook geen bladmuziek heeft, probeer dan aan een eenvoudig instrument als een blokfluit te komen, en werk de wijsjes daarop uit. U kunt een tabel maken die aangeeft welke waarde u voor welke toon op dit instrument moet gebruiken.
Tik:

```
FOR n=Ø TO 1ØØØ: BEEP .5,n: NEXT n
```

Dit geeft alle noten weer zo hoog als de computer kan gaan, en stopt daarna met een foutmelding **B integer out of range**. U kunt nu n afdrukken om te zien hoe hoog de computer kan komen.
Probeer hetzelfde, maar nu in de richting van de lage tonen. De laagste tonen klinken als klikjes. In feite bestaan de hoge tonen uit dezelfde klikjes, maar veel sneller, zodat het oor ze niet kan onderscheiden.
Alleen het middengebied van de tonen is echt geschikt voor muziek. De lagere tonen klinken te veel als klikken, en de hogere tonen klinken iel en hebben de neiging een beetje te 'zwabberen'.
Tik de volgende programma regel in:

```
1Ø BEEP .5,Ø: BEEP .5,2: BEEP .5,4: BEEP .5,5: BEEP .5,7:
   BEEP .5,9: BEEP .5,11: BEEP .5,12: STOP
```

Hierdoor speelt de computer de schaal van C groot, die alle witte toetsen van de piano tussen de midden C en een C hoger beslaat. De manier waarop deze schaal gestemd is, is precies hetzelfde als bij een piano. Dit wordt gelijkmatig stemmen genoemd, omdat het verschil in toonhoogte tussen twee halve tonen telkens gelijk is. Een violist echter zou deze toonladder net even anders spelen en alle tonen een klein beetje veranderen omdat ze dan gemakkelijker in het gehoor liggen. Hij doet dit door zijn vingers een klein beetje langs de snaren te bewegen. Bij een piano kan dit niet. De natuurlijke schaal die een violist speelt, zou er zo uit zien

```
20 BEEP .5,0: BEEP .5,2.039: BEEP .5,3.086: BEEP .5,4.98:
   BEEP .5,7.02: BEEP .5,8.84: BEEP .5,10.88: BEEP .5,12: STOP
```

Het is niet zeker dat u verschil hoort tussen deze twee regels. Sommige mensen horen het wel. Het eerste hoorbare verschil is dat de derde toon ietsje lager is in de natuurlijke schaal. Als u een echte perfectionist bent, kunt u uw programma's zo maken dat ze de natuurlijke schaal gebruiken. Het nadeel is dat deze natuurlijke schaal heel goed klinkt bij stukken die in C geschreven zijn, maar dat een stuk dat in een andere sleutel geschreven is minder mooi zal klinken - elke sleutel heeft een eigen schaal. Bij sommige stukken zal het zelfs vals klinken. De gelijkmatig gestemde schaal zit er maar een klein beetje naast, en werkt bij alle sleutels even goed.
Bij het werken met een computer is dit alles een kleiner probleem, omdat u gebruik kunt maken van de variabele **key**.
Sommige soorten muziek - bijvoorbeeld Indiase muziek - gebruikt grotere verschillen tussen de noten dan een halve toon. U kunt dit in de **BEEP** opdracht programmeren. De kwarttoon boven de midden C heeft bijvoorbeeld de waarde .5.
U kunt het toetsenbord laten piepen in plaats van klikken door

```
POKE 23609,255
```

Het tweede getal in deze opdracht geeft de lengte van de toon weer (probeer verschillende waarden tussen 0 en 255). Als het 0 is, is de toon zo kort dat het op een zachte klik lijkt.
Als u geïnteresseerd bent in meer geluidsmogelijkheden met de ZX Spectrum, zoals het horen van de tonen via een andere luidspreker dan de ingebouwde, kunt u gebruik maken van het feit dat het signaal ook aanwezig is op de 'MIC' en de 'EAR' in-/uitgangen. De 'EAR'-aansluiting geeft een iets harder signaal, maar is verder gelijk aan de MIC. U kunt hierdoor een koptelefoon aansluiten. De interne luidspreker wordt hierdoor niet uit gezet. Als u nog meer geluid wilt, kunt u de Spectrum via een van deze twee aansluitingen - de 'MIC' geeft waarschijnlijk een goed signaal - koppelen aan een externe versterker, of kunt u het geluid opnemen op een band, zodat de Spectrum zichzelf kan begeleiden.
U kunt de Spectrum hier niet mee beschadigen, zelfs niet als u de 'MIC'-en 'EAR'-aansluitingen kortsluit. Experimenteer dus zoveel u wilt.

Oefening

1. Herschrijf het Mahler-programma zodat het programma **FOR** lussen gebruikt om bepaalde stukken te herhalen.

Programmeer de computer zo dat hij niet alleen de dodenmars, maar ook de rest van Mahlers symfonie speelt.

HOOFDSTUK 20

Cassetteopslag

Samenvatting
LOAD, SAVE, VERIFY, MERGE

De fundamentele methodes voor het gebruik van een cassetterecorder om programma's te **SAVE**n, **LOAD**en en te **VERIFY**en (bewaren, laden, controleren), vindt u in deel 1 van dit boek. Dat deel moet eerst gelezen worden, en de verschillende voorschriften uitgeprobeerd, voordat hier verder gelezen wordt.
We hebben gezien dat **LOAD** het oude programma en de oude variabel wist, voordat een nieuw programma en variabelen van de cassette worden ingelezen. Er is een andere opdracht die dit niet doet; **MERGE**. **MERGE** wist alleen een programmaregel of een variabele als het niet anders kan, dat wil zeggen dat er al een programmaregel met hetzelfde nummer is, of een variabele met dezelfde naam. Tik het dobbelsteenprogramma uit hoofdstuk 11 in en bewaar het op een cassette onder de naam **dobbel**. Tik nu het volgende in, en run het vervolgens:

```
 1 PRINT 1
 2 PRINT 2
10 PRINT 10
20 LET x=20
```

ga nu verder als voor het controleren (verify), maar vervang **VERIFY "dobbel"** door

MERGE "dobbel"

Als u nu het programma list, zult u zien dat regel 1 en 2 zijn blijven staan, maar dat regel 10 en 20 vervangen zijn door de regels uit het dobbelsteenprogramma. **x** is ook blijven bestaan (probeer **PRINT x**).
U heeft nu eenvoudige voorbeelden gezien van de vier opdrachten die gebruikt worden met een cassetterecorder:
SAVE neemt het programma en de variabelen op op een cassette.
VERIFY controleert het programma en de variabelen die op de cassette staan aan de hand van het programma en de variabelen in de computer.
LOAD wist het geheugen van de computer en verwijdert daarbij het gehele programma en alle variabelen, en vervangt deze door een programma en variabelen die van een cassette worden gelezen.

MERGE werkt als **LOAD**, maar wist oude programmaregels en variabelen niet zover het nieuwe programma van cassette niet dezelfde programmaregels of variabele namen heeft.
In elk van deze gevallen wordt het sleutelwoord gevolgd door een string. Bij **SAVE** zorgt dit voor een naam van het programma op cassette, terwijl bij de andere drie de string de naam van het programma geeft waarnaar de computer op de cassette moet zoeken. Terwijl de computer aan het zoeken is, wordt de naam van elk programma dat tegengekomen wordt, weergegeven op het scherm. Er zijn enkele speciale aandachtspunten hierbij.
Bij **VERIFY, LOAD** en **MERGE** kunt u de computer een lege string geven als naam van het te zoeken programma: de computer let dan niet op de naam, maar neemt het eerste programma dat hij tegenkomt.
Een variant op **SAVE** heeft als vorm

SAVE string **LINE** nummer

Een programma dat op deze wijze bewaard wordt, is zo opgenomen dat als het programma de computer ingelezen wordt door middel van een **LOAD** (maar niet een **MERGE**), de computer onmiddellijk na het lezen naar de opgegeven regel springt, waardoor het programma zichzelf runt.
Tot nu toe hebben we alleen programma's en de gebruikte variabelen op cassette opgeslagen. Nog twee andere zaken kunnen opgeslagen worden op cassette: *arrays* en *bytes*.
Arrays worden op een iets andere wijze bewaard:
U kunt arrays op band opslaan door **DATA** te gebruiken in een **SAVE** opdracht zoals

SAVE string **DATA** array naam()

string is de naam die de verzameling gegevens op de band zal hebben. Verder werkt dit precies als bij het opnemen van een programma.
De arraynaam specificieert de array die opgenomen moet worden, dus dit bestaat uit een enkele letter, of een enkele letter gevolgd door een $. Vergeet de haakjes achter de naam niet. U zou kunnen denken dat deze logisch gezien niet nodig zijn, maar op deze manier maakt u het gemakkelijker voor de computer.
Wees duidelijk over de verschillende taken van de *string* naam en de *array* naam. Als u bijvoorbeeld zegt:

SAVE "Jan" DATA b()

zorgt de **SAVE** ervoor dat de array *b* van de computer genomen wordt en op band opgeslagen wordt onder de naam "Jan". Als u nu tikt

VERIFY "Jan" DATA b()

zal de computer zoeken naar een numerieke array die op de cassette is opgeslagen onder de naam "Jan" (als deze gevonden wordt zal op scherm afgedrukt worden: Number array: Jan) en deze vergelijken met de array *b* in de computer.

LOAD "Jan" DATA b()

zoekt de array op de cassetteband, en laadt deze - als er ruimte in het geheugen is - in het geheugen van de computer, na elke oude array met de naam *b* vernietigd te hebben.
U kunt geen **MERGE** gebruiken bij opgeslagen arrays.
U kunt karakter(string)array's op precies dezelfde wijze opslaan. Als de computer de band afzoekt en een dergelijke array vindt, schrijft hij op 'Character array:' gevolgd door de naam. Als u een karakterarray inleest, wordt niet alleen een eerdere karakterarray met dezelfde naam uit het geheugen gewist, maar ook een string met dezelfde naam.
Byteopslag wordt gebruikt voor stukken informatie zonder enige aanwijzing waarvoor de informatie gebruikt wordt - het kan een televisiebeeld zijn, een gebruiker-definieerbaar symbool, of iets dat u zelf ontwikkeld heeft. Het kan getoond worden door het woord **CODE**, bijvoorbeeld:

SAVE "tekening" CODE 16384,6912

De eenheid waarmee in het geheugen wordt opgeslagen is de *byte* (een getal tussen Ø en 255), en elke byte heeft een *adres* (hetgeen een getal tussen de Ø en 65535 is). Het eerste getal achter **CODE** is het adres van de eerste op band te zetten byte, en het tweede getal geeft het aantal bytes dat opgeslagen moet worden aan. In ons geval is 16384 het adres van de eerste byte in het weergavebestand (welke het televisiebeeld bevat) en 6912 is het aantal bytes daarin. We slaan dus een kopie van het televisiescherm op. Probeert u maar. De naam **"tekening"** werkt net zoals de namen van programma's.
Om de gegevens terug te lezen, gebruikt u

LOAD "tekening" CODE

U kunt getallen achter **CODE** zetten in de volgende vorm:

LOAD naam **CODE** start, lengte

In dit geval is *lengte* alleen een veiligheidsmaatregel; als de computer de bytes met de juiste naam op band gevonden heeft, leest hij net zo lang bytes in totdat hij er een *lengte* aantal gevonden heeft. Als er meer bytes op de band staan, negeert hij de rest, omdat het duidelijk is dat er in dat geval meer gegevens op de band staan dan u verwachtte. Dit zou ervoor kunnen zorgen dat andere informatie overschreven wordt. Als er meer gegevens zijn dan *lengte* aangeeft, meldt de computer dit met **R Tape**

loading error (band lees fout). U kunt *LENGTE* ook weglaten, zodat de computer bytes blijft lezen, ongeacht het aantal bytes.
start geeft het eerste geheugenadres aan waar de gegevens naar toe moeten. Dit hoeft niet hetzelfde adres te zijn als waarvan de gegevens op band werden gezet. Als deze twee beginadressen gelijk zijn, kunt u *start* in een **LOAD** opdracht weglaten.
CODE 16384,6912 is zo handig bij het bewaren en inlezen van een televisiebeeld dat u deze opdracht kunt vervangen door **SCREEN$**. Bijvoorbeeld

SAVE "tekening" SCREEN$
LOAD "tekening" SCREEN$

In dit uitzonderingsgeval werkt **VERIFY** niet. **VERIFY** schrijft immers de naam van hetgeen er op de band gevonden wordt op. Uiteraard wordt de naam op het beeld opgeschreven, en als dit beeld dan vergeleken wordt met het oude beeld, is dit per definitie veranderd, want de naam van het beeld is ondertussen opgeschreven. In alle andere gevallen moet u echter altijd **VERIFY** gebruiken als u **SAVE** gebruikt.
Hierna vindt u een complete samenvatting van de vier in dit hoofdstuk behandelde opdrachten.
Naam staat voor een willekeurige stringuitdrukking, en verwijst naar de op band opgeslagen informatie. Deze uitdrukking moet bestaan uit ASCII-karakters, waarvan alleen de eerste 10 gebruikt worden.
Er zijn vier soorten informatie die op band opgeslagen kunnen worden: programma en variabelen (samen), numerieke arrays, karakterarrays, en gewoon bytes.
Tijdens het zoeken van informatie met een bepaalde naam op de cassette, drukken **VERIFY, LOAD** en **MERGE** de soort en de naam van alle gevonden informatie op het scherm af. De soort informatie wordt aangegeven door 'Program:', 'Number array:', 'Character array:' of 'Bytes:'. Als *naam* een lege string was, nemen ze de eerste informatie van de juiste soort, ongeacht de naam daarvan.

SAVE
Bewaart informatie op band onder de gegeven naam. Fout F wordt opgegeven als de naam leeg is, of 11 of meer karakters heeft.
SAVE geeft altijd het bericht **Start tape, then press any key** (start de band en druk op een toets), en wacht vervolgens op het indrukken van een toets voordat er iets wordt opgeslagen.

1. Programma en variabelen:

SAVE naam **LINE** regelnummer

bewaart het programma en de variabelen op een dusdanige manier dat **LOAD** automatisch gevolgd zal worden door

GO TO regelnummer

2. Bytes:

SAVE naam **CODE** start, lengte

bewaart een *lengte* aantal bytes beginnend op het adres *start*.

SAVE naam **SCREEN$**

is gelijk aan

SAVE naam **CODE 16384,6912**

en bewaart het televisiebeeld.

3. Arrays:

SAVE naam **DATA** ()

of

SAVE naam **DATA** $ ()

bewaart de array met de naam *letter* of *letter $* (beiden hoeven niet gerelateerd te zijn aan *naam*).

VERIFY

Controleert de informatie op de band met de informatie in het geheugen. Als dit controleren niet dezelfde gegevens te zien geeft volgt de foutmelding **R Tape loading error**.

1. Programma en variabelen:

VERIFY naam

2. Bytes:

VERIFY naam **CODE** start, lengte

Als de bytes *naam* op de band in totaal langer zijn dan *lengte*, krijgt u foutmelding R. Zonder deze fout worden de bytes op de band gecontroleerd aan de hand van de bytes in het geheugen beginnend bij adres *start*.

VERIFY naam **CODE** start

controleert de bytes *naam* op band met de bytes in het geheugen, beginnend bij adres *start*.

VERIFY naam **CODE**

controleert de bytes *naam* op band met de bytes in het geheugen beginnend bij het adres vanwaar de eerste cassette byte bewaard werd.

VERIFY naam **SCREEN$**

is gelijk aan

VERIFY naam **CODE 16384,6912**

maar zal bijna zeker niet kunnen controleren.

3. Arrays

VERIFY naam **DATA** letter ()

of

VERIFY naam **DATA** letter $()

controleert de array *naam* op de band met de array *letter* of *letter $* in het geheugen.

LOAD

Laadt nieuwe informatie van de band in het geheugen. Oude informatie wordt hierbij gewist.

1. Programma en variabelen:

LOAD naam

wist het oude programma en variabelen en leest het programma en de variabelen *naam* van de cassette in. Als het programma opgeslagen werd door middel van **SAVE** naam **LINE**, volgt tevens een automatische sprong naar het regelnummer.
Foutmelding **4 Out of memory** (niet genoeg geheugenruimte) wordt gegeven als er niet genoeg ruimte in het geheugen is voor het programma en de variabelen. In dit geval worden het oude programma en de variabelen niet gewist.

2. Bytes:

LOAD naam **CODE** start, lengte

Als de bytes *naam* langer zijn dan *lengte*, volgt een foutmelding R. In andere gevallen worden de bytes van de band in het geheugen geladen, beginnend bij adres *start* daarbij wissend wat er stond.

LOAD naam **CODE** start

laadt de bytes *naam* van de band in het geheugen, beginnend bij adres *start*, daarbij het oude uitwissend.

LOAD naam **CODE**

laadt de bytes *naam* van de band in het geheugen beginnend bij het adres vanwaar de eerste byte op de band bewaard werd. De oude informatie van de geheugenplaatsen wordt gewist.

3. Arrays:

LOAD naam **DATA** letter ()

of

LOAD naam **DATA** letter $()

wist elke array met de naam *letter* of *letter $* (welke van toepassing is) en vormt een nieuwe array van de array die op band stond.
U krijgt foutmelding **4 Out of memory** als er geen geheugenruimte genoeg is voor de nieuwe array. De oude array wordt dan niet gewist.

MERGE
Laadt nieuwe informatie van cassette zonder oude informatie uit het geheugen te wissen.

1. Programma's en variabelen:

MERGE naam

voegt het programma *naam* samen met een al bestaand programma in het geheugen. Regelnummers of variabelen uit het oude programma die gelijkluidend zijn in het nieuwe programma worden uit het oude programma gewist.
U krijgt de foutmelding **4 Out of memory** als er niet genoeg geheugenruimte is voor zowel het oude programma en variabelen *als* het nieuwe programma en variabelen.

2. Bytes:

niet mogelijk

3. Arrays:

niet mogelijk

Oefeningen

1. Maak een cassette waarvan het eerste programma, als het geladen wordt, een *menu* (een lijst met alle op de cassette aanwezige programma's) afdrukt, u vraagt een keuze te maken, en vervolgens het desbetreffende programma laadt.

2. Neem de grafische schaakstukken van hoofdstuk 14, en tik daarna **NEW**. De schaakstukken zullen dit overleven. Maar het aan- en uitzetten van de computer zullen ze niet overleven. Als u ze wilt bewaren, zult u ze op cassette moeten zetten met behulp van **SAVE** en **CODE**. De eenvoudigste manier om alle 21 gebruiker-definieerbare karakters te bewaren is

SAVE "schaak" CODE USR "a",21*8

gevolgd door

VERIFY "schaak" CODE

Dit is het systeem van het bewaren van *bytes* zoals gebruikt werd om een beeld te bewaren. Het adres van de eerste te bewaren byte is **USR "a"**, het adres van de eerste van de bytes die het patroon van een gebruiker-definieerbaar karakter bepalen. Het aantal te bewaren bytes is 21*8 - acht bytes voor elk van de 21 symbolen.
Om dit alles terug in te lezen, zou u normaal gesproken gebruiken:

LOAD "schaak" CODE

Maar als u teruglaadt naar een Spectrum met een andere hoeveelheid geheugen, of als u de gebruiker-definieerbare grafieken naar een ander adres verplaatst heeft (dit is soms onontkoombaar, en gaat door middel van gevorderde technieken), moet u voorzichtiger zijn en dient u te gebruiken:

LOAD "schaak" CODE USR "a"

USR zorgt ervoor dat de grafieken naar een ander adres teruggeladen moeten worden.

HOOFDSTUK 21

De ZX-printer

Samenvatting

LPRINT, LLIST, COPY

Opmerking: Geen van bovenstaande opdrachten hoort tot standaard BASIC. **LPRINT** wordt door sommige andere computers ook wel gebruikt.

Als u een ZX Printer heeft, heeft u daar een handleiding bij gekregen. Dit hoofdstuk behandelt de BASIC-opdrachten die nodig zijn om de printer te laten werken.
De eerste twee **LPRINT** en **LLIST** zijn identiek aan **PRINT** en **LIST**, maar werken met de printer in plaats van de televisie. (De L is een historische vergissing. Toen BASIC uitgevonden werd, werd er meestal een elektrische schrijfmachine gebruikt in plaats van een beeldscherm. **PRINT** betekende dus echt 'print' (=afdrukken). Als je toen veel uitvoer had, gebruikte je een zogenaamde lineprinter. **LPRINT** betekende 'Line printer print'.)
Probeer als voorbeeld het volgende programma.

```
10 LPRINT "Dit programma"
20 LLIST
30 LPRINT '"drukt de karakterset af."'
40 FOR n=32 TO 255
50 LPRINT CHR$ n;
60 NEXT n
```

De derde opdracht **COPY**, drukt een kopie van het beeldscherm af. Tik bijvoorbeeld **LIST** om een programmalisting op het scherm te krijgen, en tik dan

```
COPY
```

Opmerking: **COPY** werkt niet bij een automatische listing, want deze wordt telkens gewist bij het opvolgen van een opdracht. U moet ofwel eerst **LIST** gebruiken, ofwel **LLIST** gebruiken en **COPY** helemaal weglaten.
U kunt de printer altijd laten stoppen door de **BREAK** toets in te drukken (**CAPS SHIFT** met **SPACE**).
Als u deze opdrachten uitvoert zonder dat de printer aangesloten is, verliest de computer alle uitvoer, en gaat hij door met de volgende opdracht.

Probeer

```
10 FOR n=31 TO 0 STEP -1
20 PRINT AT 31-n,n; CHR$ (CODE "0"+n);
30 NEXT n
```

U moet nu een patroon van karakters zien dat diagonaal van rechtsboven naar beneden loopt.
Verander nu in regel 20 **AT 31-n,n** door **TAB n**. Het programma geeft precies hetzelfde resultaat als net.
Verander nu in regel 20 **PRINT** in **LPRINT**. Deze keer zal er geen **scroll?** zijn. Met de printer komt deze vraag niet voor, en gaat de printer door met het patroon tot en met de letter O.
Verander nu **TAB n** in **AT 31-n,n**, terwijl u **LPRINT** laat staan. Nu krijgt u slechts een enkele regel met symbolen. De reden voor dit verschil is dat de uitvoer van **LPRINT** niet direkt afgedrukt wordt, maar opgeslagen wordt in een buffer in de vorm van een lange regel met door de computer af te drukken symbolen. Het feitelijke afdrukken vindt pas plaats als:
(i) de buffer vol is,
(ii) er een **LPRINT** opdracht is die niet eindigt met een komma of een puntkomma,
(iii) een komma, aanhalingsteken of **TAB** een nieuwe regel vereist, of
(iv) aan het eind van het programma, indien er dan nog iets te drukken over is.
Het gestelde onder (iii) geeft aan waarom het programma met **TAB** werkt op de manier waarop u het gezien heeft. In het geval van **AT** wordt het regelnummer genegeerd en wordt de **LPRINT** positie veranderd in het juiste kolomnummer (net als de **PRINT** positie, maar dan voor de printer). Een **AT** item kan nooit zorgen voor een nieuwe regel op de printer.

Oefening

1. Maak een afdruk van een **SIN** door het programma in hoofdstuk 17 uit te laten voeren en daarna **COPY** te gebruiken.

HOOFDSTUK 22

Verdere uitrusting

Er bestaan nog andere apparaten die u aan de Spectrum kunt koppelen.
De ZX Microdrive is een massa-opslagapparaat met hoge snelheid, en is veel flexibeler in het gebruik dan een cassetterecorder. Het werkt niet alleen met **SAVE, VERIFY, LOAD** en **MERGE**, maar ook met **PRINT, LIST, INPUT** en **INKEY$**.
Het netwerk wordt gebruikt om verschillende Spectrums met elkaar te verbinden, zodat ze met elkaar kunnen praten. Een van de voordelen hiervan is dat u bijvoorbeeld maar een microdrive nodig heeft voor verschillende Spectrums.
De RS232 interface (omschakelkastje) is een standaard verbinding die u de mogelijkheid biedt om een ZX Spectrum te verbinden met een toetsenbord, een printer, een computer of andere machines, zelfs als ze niet speciaal voor de Spectrum ontworpen zijn.
De opdrachten die u wel op het toetsenbord vindt, maar die niet gebruikt kunnen worden zonder dat deze extra apparaten aangesloten zijn, zijn o.a.: **OPEN#, CLOSE# MOVE, ERASE, CAT** en **FORMAT**

HOOFDSTUK 23

IN en OUT

Samenvatting
OUT
IN

De processor kan uit het geheugen lezen, en (bij het RAM-gedeelte) in het geheugen schrijven door middel van **PEEK** en **POKE**. De processor zelf kan het weinig schelen of een bepaalde geheugenplaats in het ROM- of het RAM-gedeelte zit, of zelfs niet bestaat. De processor weet alleen dat er 65536 geheugenplaatsen zijn, en hij kan een byte van elk adres lezen (zelfs als er onzin staat) en een byte naar elk adres schrijven (zelfs als de informatie daarmee verloren gaat). Op analoge wijze zijn er 65536 zogenaamde *I/O poorten* (Invoer/uitvoer kanalen). Deze worden door de processor gebruikt om te communiceren met bijvoorbeeld het toetsenbord of de printer. Deze poorten kunnen met behulp van BASIC beïnvloedt worden door de **IN** funktie en de **OUT** opdracht.
IN is een funktie als **PEEK**.

IN adres

IN heeft een argument, het poortadres, en het resultaat is de byte die van die poort gelezen wordt.
OUT is een opdracht als **POKE**.

OUT adres, waarde

schrijft de opgegeven waarde naar de poort met het opgegeven adres. Hoe het adres geïnterpreteerd wordt hangt af van de rest van de computer; meestal betekenen een heleboel adressen hetzelfde. Met de Spectrum is het het handigste om een adres als binair getal te zien, omdat de verschillende bits meestal onafhankelijk werken. Er zijn 16 bits in een adres. We noemen ze (met een A van *Adres)*

A15, A14, A13, A12, ..., A2, A1, AØ

Hierbij is AØ de 1-en bit, A1 de 2-en bit, A2 de 4-en bit etc. De bits AØ, A1, A2, A3 en A4 zijn de belangrijkste bits. Zij zijn normaal 1, maar als een van hen Ø is, vertelt dit de computer om iets speciaals te doen. De computer kan maar een ding tegelijkertijd doen,

dus mag nooit meer dan een van deze vijf bits op 0 staan. De bits A6 en A7 worden genegeerd, dus als u een elektronicatovenaar bent, kunt u ze zelf gebruiken. De adressen die het beste te gebruiken zijn, zijn de adressen die 1 minder zijn dan een veelvoud van 32. In dat geval zijn A0,...A4 allemaal 1. De bits A8, A9 en verder worden soms gebruikt om extra informatie door te geven.

De byte die gelezen of geschreven wordt heeft 8 bits, en deze worden meestal (met de D van *Data*) D7, D6, ..., D1, D0 genoemd. Hier vindt u een lijst met de gebruikte poortadressen.

Er is een set met inputadressen die het toetsenbord lezen en tevens de EAR-ingang. Het toetsenbord is verdeeld in 8 halve rijen van 5 toetsen elk.

IN 65278 leest de halve rij **CAPS SHIFT** tot **V**
IN 65022 leest de halve rij **A** tot **G**
IN 64510 leest de halve rij **Q** tot **T**
IN 63486 leest de halve rij **1** tot **5**
IN 61438 leest de halve rij **0** tot **6**
IN 57342 leest de halve rij **P** tot **7**
IN 49150 leest de halve rij **ENTER** tot **H**
IN 32766 leest de halve rij **SPACE** tot **B**

(Deze adressen zijn 254+256*(255-2↑n) als *n* van 0 tot 7 gaat.)

In de ingelezen byte staan de bits D0 tot D4 voor de vijf toetsen in de opgegeven halve rij - D0 voor de buitenste toets, D4 voor de toets die het dichtste bij het midden is. De bit is 0 als de toets is ingedrukt en 1 als hij niet ingedrukt is. D6 is de waarde bij de EAR-uitgang.

Poortadres 254 bestuurt bij uitvoer de luidspreker (D4) en de MIC-uitgang (D3). Tevens wordt hiermee de kleur van de border (rand) bepaald (D2, D1 en D0).

Poortadres 251 stuurt de printer, zowel voor het lezen als het schrijven: het lezen bekijkt of de printer klaar is om meer informatie te ontvangen, en schrijven stuurt de punten uit zoals ze afgedrukt moeten worden.

Poortadressen 254, 247 en 239 worden voor de extra apparaten uit hoofdstuk 22 gebruikt.

Run het volgende programma

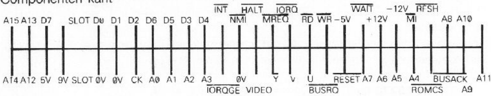
```
10 FOR n=0 TO 7: REM halve rij getal
20 LET a=254+256*(255-2↑n)
30 PRINT AT 0,0; IN a: GO TO 30
```

en speel wat met de toetsen. Als u genoeg heeft van elke halve rij tikt u **BREAK** gevolgd door

```
NEXT n
```

De controle-, data- en adresbussen zijn alle zichtbaar aan de achterkant van de Spectrum. Alles wat met een Z80 gedaan kan worden, kunt u hierdoor ook met de ZX Spectrum doen. Soms zal de Spectrum hardware in de weg zitten. Hier een schema van de aansluitingen aan de achterkant:

Componenten kant



Onderkant

HOOFDSTUK 24

Het geheugen

Samenvatting

CLEAR

Binnenin de computer wordt alles opgeslagen in de vorm van bytes, dat wil zeggen getallen tussen Ø en 255. U kunt bij uzelf denken dat u de prijs van de wol of het adres van uw boerenbond heeft opgeslagen, maar het enige dat de computer ziet zijn de bytes waarin deze informatie is omgezet.
Elke plaats waar een byte opgeslagen kan worden heeft een adres, dat een getal heeft tussen Ø en FFFFh (een adres kan opgeslagen worden als twee bytes). U kunt zich het geheugen voorstellen als een lange rij van genummerde doosjes waarbij in elk doosje een byte past. Niet alle doosjes zijn hetzelfde. In een standaard 16K RAM-machine ontbreken de doosjes van 8ØØØh tot FFFFh. De doosjes van 4ØØØh tot 7FFFh zijn RAM-doosjes hetgeen wil zeggen dat u de deksel van het doosje kunt halen, en de inhoud kunt veranderen. De doosjes van Ø tot 3FFFh zijn ROM- doosjes. Deze hebben glazen deksels die niet open kunnen. U kunt echter wel lezen wat er in de doosjes zit.

ROM	RAM	Niet gebruikt	
Ø	4ØØØh =16384	8ØØØh =32768	FFFFh =65535

Om de inhoud van een doosje te controleren gebruiken we de **PEEK** funktie: het argument is het adres van de doos, het resultaat is de inhoud ervan. Dit programma drukt bijvoorbeeld de eerste 21 bytes van het ROM geheugen (en hun adressen) af:

```
1Ø PRINT "Adres"; TAB 8; "Byte"
2Ø FOR a=Ø TO 2Ø
3Ø PRINT a; TAB 8; PEEK a
4Ø NEXT a
```

De bytes die u hiermee te zien krijgt, betekenen waarschijnlijk weinig voor u. De processorchip begrijpt ze wel, en leest eruit welke instrukties aan hem gegeven worden.
Om de inhoud van een doos te veranderen (alleen mogelijk in RAM) gebruiken we de **POKE** opdracht. Deze heeft de vorm

POKE adres, nieuwe inhoud

waarbij 'adres' en 'nieuwe inhoud' staan voor numerieke uitdrukkingen. Als u bijvoorbeeld zegt

POKE 31000,57

krijgt de byte op adres 31000 de nieuwe waarde 57. Tik

PRINT PEEK 31000

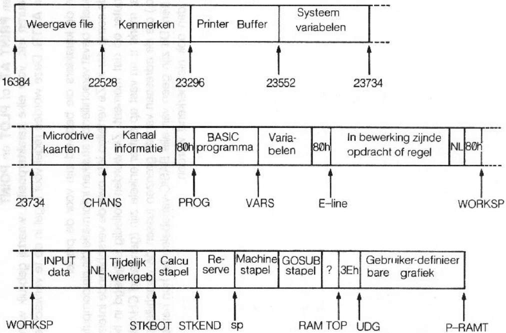
om te zien dat dit klopt. (Om uzelf te overtuigen dat we niet vals spelen kunt u ook andere adressen proberen.) De nieuwe waarde moet tussen -255 en +255 liggen. Als de waarde negatief is wordt er 256 bij opgeteld. De mogelijkheid om te poken geeft u ongekende mogelijkheden om de computer te beïnvloeden als u weet hoe u het moet gebruiken. Als u het echter verkeerd gebruikt, kunt u veel schade aanrichten. Het is erg eenvoudig door slechts een enkele verkeerde waarde op een verkeerd adres te poken, een lang en moeizaam ingetikt programma helemaal te vernielen. Gelukkig kunt u de computer hier niet mee beschadigen.
We kijken nu iets gedetaileerder naar het RAM-geheugen. Als u niet geïnteresseerd bent, kunt u dit rustig overslaan.
Het geheugen is in verschillende gebieden verdeeld (zie het grote schema) om verschillende soorten gegevens in op te slaan. De gebieden zijn net groot genoeg voor de informatie die ze bevatten. Als u meer informatie toevoegt (bijvoorbeeld door een programmaregel of een variabele toe te voegen) wordt er ruimte op het gewenste punt gemaakt, door alles boven dat punt naar boven op te schuiven. Andersom geldt dat als u informatie verwijdert, alles naar beneden geschoven wordt.
Het weergavebestand bewaart het televisiebeeld. Dit is tamelijk vreemd opgebouwd, en u zult waarschijnlijk hier niet zo snel de **PEEK** of **POKE** opdracht willen gebruiken. Elke karakterpositie op het scherm bestaat uit een 8 x 8 vierkant van lichtpunten, en elk punt kan 0 (papier) of 1 (inkt) zijn. Door gebruik te maken van binaire notatie kunnen we een patroon opslaan in 8 bytes, een byte voor elke rij. Maar deze 8 bytes worden niet bij elkaar opgeslagen. De bij elkaar horende rijen in de 32 karakters van een enkele lijn, worden opgeslagen als een rij van 32 bytes. De elektronische straal van de televisie die het beeldscherm van links naar rechts per lijn beschrijft heeft deze gegevens achter elkaar nodig om onafgebroken te kunnen schrijven. Omdat het complete beeld 24 regels van elk 8 lijnen heeft, zou u een totaal van 172 lijnen verwachten die allemaal na elkaar opgeslagen zijn. Helaas is dit niet zo. De lijnen zijn opgeslagen in blokken van telkens 8 regels. Dat wil zeggen eerst de bovenste lijn van de regels 0 tot 7, dan de twee-

de lijn van de regels Ø tot 7 enzovoorts tot de onderste lijn van de regels Ø tot 7, en daarna de bovenste lijn van de regels 8 tot 15, etc. Tot slot de lijnen van de regels 16 tot 23. Het resultaat van dit alles is dat als u gewend was aan een computer die **PEEK** en **POKE** gebruikt om het beeldscherm te beschrijven, u zult moeten wennen aan het gebruik van **SCREEN$** en **PRINT AT**, of **PLOT** en **POINT**.

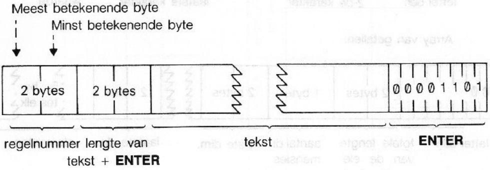
De kenmerken zijn de kleuren etc. voor elke karakterpositie, waarbij gebruik wordt gemaakt van de vorm van **ATTR**. Deze worden regel voor regel in de te verwachten volgorde opgeslagen.

De printer buffer bewaart de karakters die bedoeld zijn voor de printer.

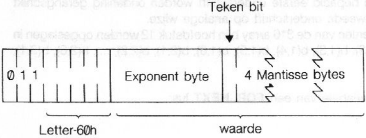
Het systeemvariabelegedeelte bevat verschillende stukken informatie die de computer vertellen 'in welke staat' de computer is. Ze vertellen iets over de verschillende interne onderdelen van de computer op dat moment. Ze worden volledig uitgelegd in het volgende hoofdstuk, maar merkt u nu vast op dat er enkele zijn (genaamd CHANS, PROG, VARS, E-LINE etc.) die de adressen van de grenzen tussen de verschillende geheugengebieden bevatten. Dit zijn geen van alle BASIC-variabelen en hun namen zullen door de computer dan ook niet herkend worden.


Weergave file
Kenmerken
Printer Buffer
Systeem variabelen
16384
22528
23296
23552
23734
Microdrive kaarten
Kanaal informatie
80h
BASIC programma
Varia-belen
80h
In bewerking zijnde opdracht of regel
NL
80h
23734
CHANS
PROG
VARS
E-line
WORKSP
INPUT data
NL
Tijdelijk 'werkgeb
Calcu stapel
Re-serve
Machine stapel
GOSUB stapel
?
3Eh
Gebruiker-definieer bare grafiek
WORKSP
STKBOT
STKEND
sp
RAM TOP
UDG
P-RAMT

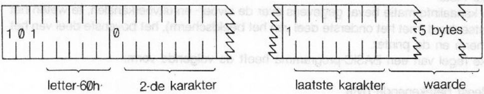
De Microdrivekaarten worden alleen gebruikt met de Microdrive. Zonder deze staat daar niets.
De kanaalinformatie bevat gegevens over de invoer- en uitvoerkanalen, te weten het toetsenbord (met het onderste deel van het beeldscherm), het bovenste deel van het scherm en de printer.
Elke regel van een BASIC-programma heeft de volgende vorm:



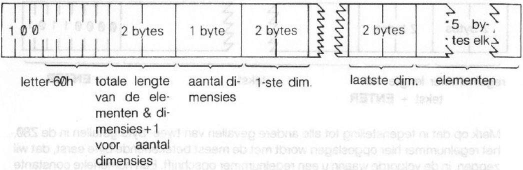
Merk op dat in tegenstelling tot alle andere gevallen van twee- byte getallen in de Z80, het regelnummer hier opgeslagen wordt met de meest betekenende byte eerst, dat wil zeggen, in de volgorde waarin u een regelnummer opschrijft. Een numerieke constante in het programma wordt gevolgd door de binaire vorm, gebruik maken van het karakter **CHR$ 14** gevolgd door 5 bytes voor het getal zelf. De variabelen hebben verschillende vormen die afhangt van de soort variabele. De letters in de namen moeten voorgesteld worden te beginnen met een kleine letter.
Getal waarvan de naam maar een letter heeft:



Getal waarvan de naam langer is dan een letter:

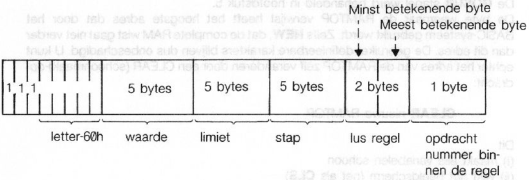


Array van getallen:

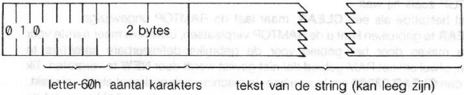


De volgorde van de elementen is:
eerst, de elementen waarvan het eerste onderschrift 1 is
dan, de elementen waarvan het eerste onderschrift 2 is
dan, de elementen waarvan het eerste onderschrift 3 is
en zo verder voor alle voorkomende waarden van het eerste onderschrift.
De elementen met een bepaald eerste onderschrift worden onderling gerangschikt aan de hand van het tweede onderschrift op analoge wijze.
Als voorbeeld: de elementen van de 3*6 array b in hoofdstuk 12 worden opgeslagen in de volgorde b(1,1), b(1,2), b(1,3), b(1,4), b(1,5), b(1,6), b(2,1), b(2,2), ..., b(2,6), b(3,1), b(3,2), ..., b(3,6).

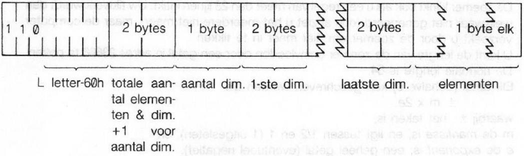
De controlevariabele van een **FOR..NEXT** lus:



String:



Array met karakters:



De calculator is het deel van het BASIC-systeem dat de rekenkunde behandelt. De getallen waarmee gewerkt wordt, worden meestal opgeslagen in de calculatorstapel.
Het reservedeel bevat de ruimte die tot nu toe ongebruikt is gebleven.
De machinestapel is de stapel die de Z80 processor gebruikt om onder andere terugkeeradressen in op te slaan.

De **GOSUB** stapel werd behandeld in hoofdstuk 5.
De byte waarnaar de RAMTOP verwijst heeft het hoogste adres dat door het BASIC-systeem gebruikt wordt. Zelfs **NEW**, dat de complete RAM wist gaat niet verder dan dit adres. De gebruiker-definieerbare karakters blijven dus onbeschadigd. U kunt echter het adres van de RAMTOP zelf veranderen door een CLEAR (schoonmaak)-opdracht:

CLEAR nieuwe RAMTOP

Dit
(i) maakt alle variabelen schoon
(ii) wist het beeldscherm (net als **CLS**)
(iii) zet de **PLOT** positie terug op de linkerbenedenhoek
(iv) zorgt voor een **RESTORE**
(v) leegt de **GOSUB** stapel en plaats deze bij de nieuwe RAMTOP - ervan uitgaande dat deze tussen de calculatorstapel en het feitelijke einde van de RAM ligt. Anders blijft de RAMTOP zoals hij was.
RUN doet hetzelfde als een **CLEAR**, maar laat de RAMTOP ongewijzigd.
Door **CLEAR** te gebruiken kunt u de RAMTOP verplaatsen, ofwel om meer ruimte voor BASIC te maken door het gebied voor de gebruiker-definieerbare karakters te gebruiken, ofwel om het RAM-gebied dat niet gewist wordt door **NEW** te vergroten. Tik **NEW** en dan **CLEAR 23800** om te zien hoe de machine zich gedraagt als hij vol raakt.
Een van de eerste dingen die u bemerkt als u een programma in gaat tikken is dat de computer na een tijdje geen invoer meer accepteert en gaat zoemen. Dit betekent dat de computer boordevol zit en dat u hem voorzichtig moet legen. Er zijn twee foutmeldingen die ongeveer hetzelfde betekenen, **4 Memory full** (geheugen vol) en **G No room for line** (geen ruimte voor de regel).
De zoemer klinkt ook als u een regel van meer dan 23 lijnen intikt. Uw tikwerk wordt dan weliswaar niet genegeerd (ook al ziet u het meerdere niet meer), maar de computer verzoekt u door de zoemer om niet meer in te tikken.
U kunt de lengte van de zoemer beïnvloeden door een getal in adres 23608 te poken. De normale lengte is 64.
Elk getal (behalve 0) kan geschreven worden als

$\pm$ m x 2e

waarbij $\pm$ het teken is,
m de *mantisse* is, en ligt tussen 1/2 en 1 (1 uitgesloten), en
e de *exponent* is; een geheel getal (eventueel negatief).
Stel dat u m in het binaire stelsel wilt schrijven. Omdat het een breuk is, heeft het een *binaire punt* (net als de decimale punt a bij getallen in het 10-tallige stelsel) en vervolgens een binaire fractie (net als een decimale fractie): in binair wordt
een half geschreven als .1
een kwart geschreven als .01
drie kwart geschreven als .11
een tiende geschreven als 0.. etc. Voor onze m geldt dat, omdat het minder is dan 1, er

geen bits voor de binaire punt staan, en omdat het meer dan een half is, de eerste bit achter de binaire punt een 1 is.
Om het getal in de computer op te slaan, gebruiken we vijf bytes, als volgt:
(i) schrijf de eerste 8 bits van de mantisse in de tweede byte (we weten dat de eerste bit 1 is), de volgende 8 bits in de derde byte, de volgende 8 bits in de vierde byte en de laatste 8 bits in de vijfde byte,
(ii) vervang de eerste bit in de tweede byte - waarvan we weten dat hij 1 is - door het teken: 0 voor positief, 1 voor negatief,
(iii) schrijf de exponent +128 in de eerste byte.
Bijvoorbeeld het getal 1/10:

$$1/10 = 4/5 \times 2^{-3}$$

De mantisse m is dus: EC011001100 C110011001100 binair (omdat de 33-ste bit 1 is, ronden we de 32-ste bit af van 0 naar 1). De exponent e is -3. Als we de drie regels van hierboven toepassen, krijgen we

Hier een nul om een + teken aan te geven
↓

0111 1101	0100 1100	1100 1100	1100 1100	1100 1101

-3 + 128

mantisse 4/5 maar de 1-ste bit zou 1 moeten zijn voor de exponent

Er is nog een alternatieve manier om gehele getallen tussen - 65535 en +65535 op te slaan:
(i) de eerste byte is 0
(ii) de tweede byte is 0 voor een positief getal, FFh voor een negatief getal
(iii) de derde en vierde byte zijn de minst en de meest betekenende byte van het getal (of het getal +131072 als het een negatief getal is),
(iv) de vijfde byte is 0.

HOOFDSTUK 25

De systeemvariabelen

De bytes in het geheugen van 23552 en 23733 zijn apart gezet voor speciale taken voor het systeem. U kunt door middel van **PEEK** hun inhoud bekijken om daarmee allerlei zaken over de toestand van het systeem te weten te komen. Enkelen ervan kunt u beïnvloeden door middel van **POKE**. Ze worden hier, samen met het gebruik ervan opgesomd.
Deze geheugenbytes worden systeemvariabelen genoemd. Hoewel ze namen hebben, moeten ze niet verward worden met de variabelen die gebruikt worden in BASIC. De computer zal de systeemvariabelen niet herkennen als u ze bij naam noemt. De namen zijn zogenaamde mnemonics. Dit zijn afkortingen. Helaas voor de Nederlandse lezer gaat het om afkortingen van de Engelstalige omschrijvingen van de verschillende taken van de systeemvariabelen. Omdat deze mnemonics in alle literatuur altijd op dezelfde wijze gebruikt wordt, zijn deze niet vertaald. Waar nodig is een Nederlandse omschrijving gegeven.
De afkortingen in kolom 1 hebben de volgende betekenissen:

X - De variabele moet niet in het geheugen geschreven worden door **POKE** omdat daardoor het systeem vast kan lopen.
N - In het geheugen schrijven door **POKE** heeft geen blijvend effect.
Het getal in kolom 1 geeft het aantal bytes in de variabele aan. Bij twee bytes is de eerste de minst betekenende byte - het tegenovergestelde van wat u zou verwachten. Om een waarde **v** aan een twee-byte variabel op adres **n** toe te kennen gebruikt u:

```
POKE n,v-256*INT(v/256)
POKE n+1,INT(v/256)
```

Om de waarde te lezen gebruikt u:

```
PEEK n+256*PEEK (n+1)
```

Opmerking	Adres	Naam	Inhoud
N8	23552	KSTATE	Gebruikt om het toetsenbord te lezen.
N1	23560	LAST K	Slaat de recent ingedrukte toets op.
1	23561	REPDEL	De tijd (in 50-sten van een seconde, 60-sten in USA) dat een toets ingedrukt moet worden voordat deze zichzelf herhaald. Standaardwaarde is 35, maar u kunt andere waarden door **POKE** gebruiken.
1	23562	REPPER	De vertraging (in 50-sten of 60-sten van een seconde) tussen twee herhalingen van een ingedrukte toets. Standaardwaarde 5.
N2	23563	DEFADD	Adres van de argumenten van een gebruiker-definieerbare funktie, als er zo'n funktie gedefinieerd is. Anders 0.
N1	23565	K DATA	Opslag van de 2-de byte van de kleurcontrole door het toetsenbord ingevoerd.
N2	23566	TVDATA	Opslag van bytes voor kleur, **AT** en **TAB** controles die naar de televisie gaan.
X38	23568	STRMS	Adressen van de kanalen die aan de verschillende stromen zijn gekoppeld.
2	23606	CHARS	256 minder dan het adres van de karakterset (die start met een spatie en loopt tot het copyrightsymbool). Normaal gesproken zit dit in ROM, maar u kunt uw eigen RAM serie maken en CHARS er naar laten verwijzen.
1	23608	RASP	Lengte van de waarschuwingszoemer.
1	23609	PIP	Lengte van de toetsenbordklik.
1	23610	ERR NR	1 minder dan de rapportagecode. Begint bij 255 (voor -1) dus **PEEK 23610** geeft 255.
X1	23611	FLAGS	Verschillende vlaggen om het BASIC-systeem te besturen.
X1	23612	TV FLAG	Vlag ten behoeve van de televisie.
X2	23613	ERR SP	Adres van het item op de machinestapel dat gebruikt wordt als fout terugkeeradres.
N2	23615	LIST SP	Adres voor het terugkeeradres van een automatische listing.
N1	23617	MODE	Bepaalt K-, L-, C-, E- of G-cursor.
2	23618	NEWPPC	Regel waarnaar gesprongen moet worden.
1	23620	NSPPC	Opdrachtnummer in de regel waarnaar gesprongen moet worden. Door eerst NEWPPC en daarna NSPPC te poken veroorzaakt u een sprong naar een bepaalde opdracht in een regel.

Opmerking	*Adres*	*Naam*	*Inhoud*
2	23621	PPC	Regelnummer van de opdracht die op dit moment wordt uitgevoerd.
1	23623	SUBPPC	getal binnen de regel van de opdracht die op dit moment wordt uitgevoerd.
1	23624	BORDCR	Borderkleur (rand) * 8; bevat tevens de kenmerken die normaal gesproken gebruikt worden voor het onderste deel van het scherm.
2	23625	E PPC	Nummer van de tegenwoordige (current) regel (met programma cursor).
X2	23627	VARS	Adres van de variabelen.
N2	23629	DEST	Adres van de variabele waaraan toegekend wordt.
X2	23631	CHANS	Adres van kanaalgegevens.
X2	23633	CURCHL	Adres van de informatie die op dit moment gebruikt wordt voor invoer en uitvoer.
X2	23635	PROG	Adres van BASIC-programma.
X2	23637	NXTLIN	Adres van de volgende regel in het programma.
X2	23639	DATADD	Adres van de afsluiter van het laatste DATA-item.
X2	23641	E LINE	Adres van de op dit moment ingetikte opdracht.
2	23643	K CUR	Adres van de cursor.
X2	23645	CH ADD	Adres van het eerstvolgende te interpreteren karakter: het karakter achter het argument van **PEEK** of de **NEWLINE** aan het einde van een **POKE** opdracht.
2	23647	X PTR	Adres van het karakter achter het ? karakter.
X2	23649	WORKSP	Adres van de tijdelijke werkruimte.
X2	23651	STKBOT	Adres van de bodem van de calculalorstapel.
X2	23653	STKEND	Beginadres van de reserveruimte.
N1	23655	BREG	b register van de calculator.
N2	23656	MEM	Adres van het gebied dat gebruikt wordt voor het geheugen van de calculator. (Meestal, maar niet altijd, gelijk aan MEMBOT.)
1	23658	FLAGS2	Meer vlaggen.
X1	23659	DF SZ	Aantal regels (inclusief een blanko regel) in het onderste deel van het scherm.
2	23660B	S TP	Regelnummer van de bovenste programmaregel bij automatische listing.
2	23662	OLDPPC	Regelnummer waar **CONTINUE** naar springt.
1	23664	OSPCC	Opdrachtnummer binnen de regel waar **CONTINUE** naar springt.
N1	23665	FLAGX	Verschillende vlaggen.
N2	23666	STRLEN	Lengte van stringbestemming in een toekenning.

Opmerking	Adres	Naam	Inhoud
N2	23668	T ADDR	Adres van het volgende item in een syntax (grammatika)-tabel. (zeer waarschijnlijk niet nuttig).
2	23670	SEED	De basis van **RND**. Dit is de variabele die gesteld wordt door **RANDOMIZE**.
3	23672	FRAMES	3 bytes (minst betekenende byte eerst) beeld teller. Wordt elke 20 microseconde verhoogd. Zie hoofdstuk 18.
2	23675	UDG	Adres van eerste gebruiker-definieerbare grafisch symbool. U kunt dit veranderen om meer ruimte te krijgen door minder gebruiker-definieerbare karakters te gebruiken.
1	23677	COORDS	x-coördinaat van het laatste getekende punt.
1	23678		y-coördinaat van het laatste getekende punt.
1	23679	P POSN	33-koloms getal van de printerpositie.
1	23680	PR CC	Minst betekenende byte van het adres van de volgende positie voor **LPRINT** om af te drukken (in printer buffer).
1	23681		Niet gebruikt.
2	23682	ECHO E	33-koloms getal en 24-regel getal (onderste deel van scherm) van het einde van de INPUT-buffer.
2	23684	DF CC	Adres in weergavebestand in **PRINT** positie.
2	23686	DFCCL	Als DF CC maar voor onderste deel van scherm.
X1	23688	S POSN	33-koloms getal voor **PRINT** positie.
X1	23689		24-regel getal voor **PRINT** positie.
X2	23690	SPOSNL	Als S POSN maar voor onderste deel van het scherm.
1	23692	SCR CT	Telt het aantal 'scrolls'. Altijd 1 meer dan het aantal scrolls dat gedaan zal zijn als er gestopt wordt met **scroll?** Als u dit adres blijft vullen met een waarde groter dan 1 (bijvoorbeeld 255), zal het beeld blijven rollen zonder u iets te vragen.
1	23693	ATTR P	Permanente tegenwoordige kleuren, etc. (zoals gesteld door de kleuropdrachten).
1	23694	MASK P	Gebruikt voor transparante kleuren, etc. Elke bit die 1 is toont dat de corresponderende kenmerkbit niet komt van ATTR P, maar van datgene dat op het scherm staat.

Opmerking	*Adres*	*Naam*	*Inhoud*
N1	23695	ATTR T	Tijdelijke tegenwoordige kleuren, etc. (zoals gesteld door de kleuritems).
N1	23696	MASK T	Als MASK P, maar tijdelijk.
1	23697	P FLAG	Nog meer vlaggen.
N30	23698	MEMBOT	Geheugengebied van de calculator; gebruikt om getallen in op te slaan die niet eenvoudig op de calculatorstapel passen.
2	23728		Niet gebruikt.
2	23730	RAMTOP	Adres van de laatste byte van het BASIC-systeemgebied.
2	23732	P-RAMT	Adres van de laatste byte van de werkelijke RAM (fysische RAM).

Dit programma geeft u de eerste 22 bytes van het variabelengebied:

```
10 FOR n=0 TO 21
20 PRINT PEEK (PEEK 23627+256*PEEK 23628+n)
30 NEXT n
```

Probeer nu de controlevariabele **n** te vergelijken met de beschrijvingen hiervoor. Verander regel 20 in

```
20 PRINT PEEK (23755+n)
```

Dit laat u de eerste 22 bytes van het programmagebied zien. Vergelijk deze met het programma zelf.

HOOFDSTUK 26

Machinecode gebruiken

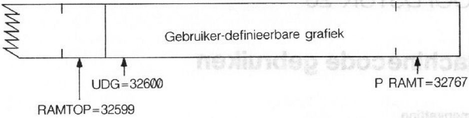
Samenvatting
USR met numeriek argument

Dit hoofdstuk is geschreven voor hen die *Z80 machinecode*, de instruktieset die de Z80 processor chip gebruikt, begrijpen. Als u dit nog niet kunt, maar wel zou willen, kunt u een van de vele boeken over dit onderwerp lezen. De titels van deze boeken luiden allemaal ongeveer 'Z80 machinecode (of machinetaal) voor de beginner', of in het Engels 'Z80 machine code (of assembly language) for the absolute beginner'. Als uit de titel blijkt dat het boek speciaal voor de ZX Spectrum is geschreven, maakt dit alles nog eenvoudiger.
Deze programma's worden normaal gesproken in *assembleertaal* geschreven, hetgeen, hoewel nogal cryptisch, met wat oefening niet zo heel moeilijk te begrijpen is. (U kunt de machinetaalinstrukties in bijlage A vinden.) Maar om deze programma's door de computer uit te laten voeren, moet het programma omgezet worden in een opeenvolging van bytes. In deze vorm wordt het programma geschreven in *machinecode*. Dit vertalen gebeurt meestal door de computer, die daarvoor een *assembleer*-programma gebruikt. De Spectrum heeft geen assembler ingebouwd, maar u kunt er een kopen op een cassette. Anders zult u het vertalen zelf moeten doen, hetgeen alleen mogelijk is als het programma niet te lang is.
Laten we als voorbeeld het volgende programma nemen

```
ld bc, 99
ret
```

Dit programma laadt het bc registerpaar met 99. Dit wordt vertaald in vier machinecode bytes; 1, 99, 0 (voor ld bc, 99) en 201 (voor ret). (Als u 1 en 201 opzoekt in bijlage A, vindt u ld bc, NN. NN staat hierbij voor een willekeurig twee-byte getal. En ret.)
Als u uw machinecodeprogramma heeft, is de volgende stap het programma in de computer te krijgen. (Een assembleerprogramma doet dit meestal automatisch.) U moet daarvoor beslissen waar in het geheugen u het programma wilt opslaan. De beste plaats is in een extra door u te maken ruimte tussen het BASIC-gebied en het gebruiker-definieerbare grafiek gebied.

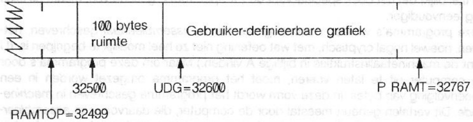
Stel dat u een 16K Spectrum heeft. Als u begint, heeft de top van de RAM:



Als u nu tikt

CLEAR 32499

krijgt u een ruimte van 100 (lekker ruim) bytes, beginnend op adres 32500.



Om het machinecodeprogramma in te voeren, kunt u een BASIC-programma maken:

```
10 LET a=32500
20 READ n:POKE a,n
30 LET a=a+1: GO TO 20
40 DATA 1,99,0,201
```

(Dit zal stoppen met de foutmelding **E Out of DATA** (zonder gegevens) zodra de vier door u opgegeven bytes gevuld zijn.)
Om het machinecodeprogramma uit te laten voeren, kunt u de funktie **USR** gebruiken, maar met een numeriek argument; het beginadres. Het resultaat is de waarde van het bc register op het moment dat het machinecodeprogramma terugkeert. Als u dus tikt

PRINT USR 32500

krijgt u als antwoord 99.
Het terugkeeradres naar de BASIC wordt op de normale manier opgestapeld. Terugkeren gaat door een Z80 ret opdracht. U dient de iy en i registers in een machinecoderoutine niet te gebruiken.

U kunt uw machinecodeprogramma eenvoudig bewaren met

```
SAVE "een naam" CODE 32500,4
```

Er is geen enkele manier om dit op een zodanige manier te bewaren dat het programma automatisch gerund wordt als het geladen is. U kunt dit echter wel simuleren door dit BASIC-programma te gebruiken

```
10 LOAD "" CODE 32500,4
20 PRINT USR 32500
```

Tik eerst

```
SAVE "een naam" LINE
```

en daarna

```
SAVE "xxxx" CODE 32500,4
LOAD "een naam"
```

Op deze wijze wordt het BASIC-programma geladen en automatisch gerund, en het BASIC-programma laadt en runt het machinecodeprogramma.

BIJLAGE A

De karakterset

Dit is de complete Spectrum karakterset, met de codes in decimale en hexadecimale notering. Als de codes voorgesteld worden als Z8Ø machinecode-instructies geven de rechterkolommen de overeenkomende machinetaal mnemonics. Diegenen die hier mee bekend zijn, weten dat sommige Z8Ø instrukties samenstellingen zijn die beginnen met CBh of EDh; De twee rechterkolommen geven deze samenstellingen.

Code	*Karakter*	*Hex*	*Z8Ø Assembler*	*-na CB*	*-na ED*
Ø	niet gebruikt	ØØ	nop	rlc b	
1	niet gebruikt	Ø1	ld bc,NN	rlc c	
2	niet gebruikt	Ø2	ld (bc),a	rlc d	
3	niet gebruikt	Ø3	inc bc	rlc e	
4	niet gebruikt	Ø4	inc b	rlc h	
5	niet gebruikt	Ø5	dec b	rlc l	
6	**PRINT** komma	Ø6	ld b,N	rlc (hl)	
7	**EDIT**	Ø7	rlca	rlc a	
8	cursor links	Ø8	ex af,af'	rrc b	
9	cursor rechts	Ø9	add hl,bc	rrc c	
1Ø	cursor omlaag	ØA	ld a,(bc)	rrc d	
11	cursor omhoog	ØB	dec bc	rrc e	
12	**DELETE**	ØC	inc c	rrc h	
13	**ENTER**	ØD	dec c	rrc l	
14	getal	ØE	ld c,N	rrc(hl)	
15	niet gebruikt	ØF	rrca	rrca	
16	**INK** besturing	1Ø	djnz DIS	rl b	
17	**PAPER** besturing	11	ld de,NN	rl c	
18	**FLASH** besturing	12	ld (de),a	rl d	
19	**BRIGHT** besturing	13	inc de	rl e	
2Ø	**INVERSE** besturing	14	inc d	rl h	
21	**OVER** besturing	15	dec d	rl l	
22	**AT** besturing	16	ld d,N	rl (hl)	
23	**TAB** besturing	17	rla	rl a	

Code	*Karakter*	*Hex*	*Z80 Assembler*	*-na CB*	*-na ED*
24		18	jr DIS	rr b	
25		19	add hl,de	rr c	
26		1A	ld a,(de)	rr d	
27		1B	dec de	rr e	
28	niet gebruikt	1C	inc e	rr h	
29		1D	dec e	rr l	
30		1E	ld e,N	rr (hl)	
31		1F	rra	rr a	
32	spatie	20	jr nz,DIS	sla b	
33	!	21	ld hl,NN	sla c	
34	"	22	ld (NN),hl	sla d	
35	#	23	inc hl	sla e	
36	$	24	inc h	sla h	
37	%	25	dec h	sla l	
38	&	26	ld h,N	sla (hl)	
39	'	27	daa	sla a	
40	(	28	jr z,DIS	sra b	
41	)	29	add hl,hl	sra c	
42	*	2A	ld hl,(NN)	sra d	
43	+	2B	dec hl	sra e	
44	,	2C	inc l	sra h	
45	-	2D	dec l	sra l	
46	.	2E	ld l,N	sra (hl)	
47	/	2F	cpl	sra a	
48	0	30	jr nc,DIS		
49	1	31	ld sp,NN		
50	2	32	ld (NN),a		
51	3	33	inc sp		
52	4	34	inc (hl)		
53	5	35	dec (hl)		
54	6	36	ld (hl),N		
55	7	37	scf		
56	8	38	jr c,DIS	srl b	
57	9	39	add hl,sp	srl c	
58	:	3A	ld a,(NN)	srl d	
59	;	3B	dec sp	srl e	
60	<	3C	inc a	srl h	
61	=	3D	dec a	srl l	
62	>	3E	ld a,N	srl (hl)	
63	?	3F	ccf	srl a	
64	@	40	ld b,b	bit 0,b	in b,(c)
65	A	41	ld b,c	bit 0,c	out (c),b
66	B	42	ld b,d	bit 0,d	sbc hl,bc

Code	Karakter	Hex	Z80 Assembler	- na CB	-na ED
67	C	43	ld b,e	bit 0,e	ld (NN),bc
68	D	44	ld b,h	bit 0,h	neg
69	E	45	ld b,l	bit 0,l	retn
70	F	46	ld b,(hl)	bit 0,(hl)	im 0
71	G	47	ld b,a	bit 0,a	ld i,a
72	H	48	ld c,b	bit 1,b	in c,(c)
73	I	49	ld c,c	bit 1,c	out (c),c
74	J	4A	ld c,d	bit 1,d	adc hl,bc
75	K	4B	ld c,e	bit 1,e	ld bc,(NN)
76	L	4C	ld c,h	bit 1,h	
77	M	4D	ld c,l	bit 1,l	reti
78	N	4E	ld c,(hl)	bit 1,(hl)	
79	O	4F	ld c,a	bit 1,a	ld r,a
80	P	50	ld d,b	bit 2,b	in d,(c)
81	Q	51	ld d,c	bit 2,c	out (c),d
82	R	52	ld d,d	bit 2,d	sbc hl,de
83	S	53	ld d,e	bit 2,e	ld (NN),de
84	T	54	ld d,h	bit 2,h	
85	U	55	ld d,l	bit 2,l	
86	V	56	ld d,(hl)	bit 2,(hl)	im 1
87	W	57	ld d,a	bit 2,a	la a,i
88	X	58	ld e,b	bit 3,b	in e,(c)
89	Y	59	ld e,c	bit 3,c	out (c),e
90	Z	5A	ld e,d	bit 3,d	adc hl,de
91	[	5B	ld e,e	bit 3,e	ld de,(NN)
92	/	5C	ld e,h	bit 3,h	
93	]	5D	le d,l	bit 3,l	
94	↑	5E	ld e,(hl)	bit 3,(hl)	im 2
95	_	5F	ld e,a	bit 3,a	ld a,r
96	£	60	ld h,b	bit 4,b	in h,(c)
97	a	61	ld h,c	bit 4,c	out (c),h
98	b	62	ld h,d	bit 4,d	sbc hl,hl
99	c	63	ld h,e	bit 4,e	ld (NN),hl
100	d	64	ld h,h	bit 4,h	
101	e	65	ld h,l	bit 4,l	
102	f	66	ld h,(hl)	bit 4,(hl)	
103	g	67	ld h,a	bit 4,a	rrd
104	h	68	ld l,b	bit 5,b	in l,(c)
105	i	69	ld l,c	bit 5,c	out (c),l
106	j	6A	ld l,d	bit 5,d	adc hl,hl
107	k	6B	ld l,e	bit 5,e	ld hl,(NN)
108	l	6C	ld l,h	bit 5,h	

Code	Karakter	Hex	Z80 Assembler	-na CB	-na ED
109	m	6D	ld l,l	bit 5,l	
110	n	6E	ld l,(hl)	bit 5,(hl)	
111	o	6F	ld l,a	bit 5,a	rld
112	p	70	ld (hl),b	bit 6,b	in f,(c)
113	q	71	ld (hl),c	bit 6,c	
114	r	72	ld (hl),d	bit 6,d	sbc hl,sp
115	s	73	ld (hl),e	bit 6,e	ld (NN),sp
116	t	74	ld (hl),h	bit 6,h	
117	u	75	ld(hl),l	bit 6,l	
118	v	76	halt	bit 6,(hl)	
119	w	77	la (hl),a	bit 6,a	
120	x	78	ld a,b	bit 7,b	in a,(c)
121	y	79	ld a,c	bit 7,c	out (c),a
122	z	7A	la a,d	bit 7,d	adc hl,sp
123	{	7B	ld a,e	bit 7,e	ld sp,(NN)
124	|	7C	ld a,h	bit 7,h	
125	}	7D	la a,l	bit 7,l	
126	-	7E	ld a,(hl)	bit 7,(hl)	
127	©	7F	ld a,a	bit 7,a	
128		80	add a,b	res 0,b	
129		81	add a,c	res 0,c	
130		82	add a,d	res 0,d	
131		83	add a,e	res 0,e	
132		84	add a,h	res 0,h	
133		85	add a,l	res 0,l	
134		86	add a,(hl)	res 0,(hl)	
135		87	add a,a	res 0,a	
136		88	adc a,c	res 1,b	
137		89	adc a,c	res 1,c	
138		8A	adc a,d	res 1,d	
139		8B	adc a,e	res 1,e	
140		8C	adc a,h	res 1,h	
141		8D	adc a,l	res 1,l	
142		8E	adc a,(hl)	res 1,(hl)	
143		8F	adc a,a	res 1,a	

Code	*Karakter*	*Hex*	*Z80 Assembler*	*- na CB*	*-na ED*
144	(a)	90	sub b	res 2,b	
145	(b)	91	sub c	res 2,c	
146	(c)	92	sub d	res 2,d	
147	(d)	93	sub e	res 2,e	
148	(e)	94	sub h	res 2,h	
149	(f)	95	sub l	res 2,l	
150	(g)	96	sub (hl)	res 2,(hl)	
151	(h)	97	sub a	res 2,a	
152	(i)	98	sbc a,b	res 3,b	
153	(j) gebruiker-	99	sbc a,c	res 3,c	
154	(k) definieerbare	9A	sbc a,d	res 3,d	
155	(l) grafiek	9B	sbc a,e	res 3,e	
156	(m)	9C	sbc a,h	res 3,h	
157	(n)	9D	sbc a,l	res 3,l	
158	(o)	9E	sbc a,(hl)	res 3,(hl)	
159	(p)	9F	sbc a,a	res 3,a	
160	(q)	A0	and b	res 4,b	ldi
161	(r)	A1	and c	res 4,c	cpi
162	(s)	A2	and d	res 4,d	ini
163	(t)	A3	and e	res 4,e	outi
164	(u)	A4	and h	res 4,h	
165	**RND**	A5	and l	res 4,l	
166	**INKEY$**	A6	and (hl)	res 4,(hl)	
167	**PI**	A7	and a	res 4,a	
168	**FN**	A8	xor b	res 5,b	ldd
169	**POINT**	A9	xor c	res 5,c	cpd
170	**SCREEN$**	AA	xor d	res 5,d	ind
171	**ATTR**	AB	xor e	res 5,e	outd
172	**AT**	AC	xor h	res 5,h	
173	**TAB**	AD	xor l	res 5,l	
174	**VAL$**	AE	xor (hl)	res 5,(hl)	
175	**CODE**	AF	xor a	res 5,a	
176	**VAL**	B0	or b	res 6,b	ldir
177	**LEN**	B1	or c	res 6,c	cpir
178	**SIN**	B2	or d	res 6,d	inir
179	**COS**	B3	or e	res 6,e	otir
180	**TAN**	B4	or h	res 6,h	
181	**ASN**	B5	or l	res 6,l	
182	**ACS**	B6	or (hl)	res 6,(hl)	
183	**ATN**	B7	or a	res 6,a	
184	**LN**	B8	cp b	res 7,b	lddr
185	**EXP**	B9	cp c	res 7,c	cpdr
186	**INT**	BA	cp d	res 7,d	indr

Code	Karakter	Hex	Z80 Assembler	- na CB	-na ED
187	**SQR**	BB	cp e	res 7,e	otdr
188	**SGN**	BC	cp h	res 7,h	
189	**ABS**	BD	cp l	res 7,l	
190	**PEEK**	BE	cp (hl)	res 7,(hl)	
191	**IN**	BF	cp a	res 7,a	
192	**USR**	C0	ret nz	set 0,b	
193	**STR$**	C1	pop bc	set 0,c	
194	**CHR$**	C2	jp nz,NN	set 0,d	
195	**NOT**	C3	jp NN	set 0,e	
196	**BIN**	C4	call nz,NN	set 0,h	
197	**OR**	C5	push bc	set 0,l	
198	**AND**	C6	add a,N	set 0,(hl)	
199	**< =**	C7	rst 0	set 0,a	
200	**> =**	C8	ret z	set 1,b	
201	**< >**	C9	ret	set 1,c	
202	**LINE**	CA	jp z,NN,set 1,d		
203	**THEN**	CB		set 1,e	
204	**TO**	CC	call z,NN	set 1,h	
205	**STEP**	CD	call NN,set 1,l		
206	**DEF FN**	CE	adc a,N	set 1,(hl)	
207	**CAT**	CF	rst 8	set 1,a	
208	**FORMAT**	D0	ret nc	set 2,b	
209	**MOVE**	D1	pop de	set 2,c	
210	**ERASE**	D2	jp nc,NN	set 2,d	
211	**OPEN#**	D3	out (N),a	set 2,e	
212	**CLOSE#**	D4	call nc,NN	set 2,h	
213	**MERGE**	D5	push de	set 2,l	
214	**VERIFY**	D6	sub N	set 2,(hl)	
215	**BEEP**	D7	rst 16	set 2,a	
216	**CIRCLE**	D8	ret c	set 3,b	
217	**INK**	D9	exx	set 3,c	
218	**PAPER**	DA	jp c,NN	set 3,d	
219	**FLASH**	DB	in a,(N)	set 3,e	
220	**BRIGHT**	DC	call c,NN	set 3,h	
221	**INVERSE**	DD	gaat vooraf aan instrukties met ix	set 3,l	
222	**OVER**	DE	sbc a,N	set 3,(hl)	
223	**OUT**	DF	rst 24	set 3,a	
224	**LPRINT**	E0	ret po	set 4,b	
225	**LLIST**	E1	pop hl	set 4,c	
226	**STOP**	E2	jp po,NN	set 4,d	
227	**READ**	E3	ex (sp),hl	set 4,e	
228	**DATA**	E4	call po,NN	4,h	

Code	Karakter	Hex	Z80 Assembler	- na CB	-na ED
229	**RESTORE**	E5	push hl	set 4,l	
230	**NEW**	E6	and N	set 4,(hl)	
231	**BORDER**	E7	rst 32	set 4,a	
232	**CONTINUE**	E8	ret pe	set 5,b	
233	**DIM**	E9	jp (hl)	set 5,c	
234	**REM**	EA	jp pe,NN	set 5,d	
235	**FOR**	EB	ex de,hl	set 5,e	
236	**GO TO**	EC	call pe,NN	set 5,h	
237	**GO SUB**	ED		set 5,l	
238	**INPUT**	EE	xor N	set 5,(hl)	
239	**LOAD**	EF	rst 40	set 5,a	
240	**LIST**	F0	ret p	set 6,b	
241	**LET**	F1	pop af	set 6,c	
242	**PAUSE**	F2	jp p,NN	set 6,d	
243	**NEXT**	F3	di	set 6,e	
244	**POKE**	F4	call p,NN	set 6,h	
245	**PRINT**	F5	push af	set 6,l	
246	**PLOT**	F6	or N	set 6,(hl)	
247	**RUN**	F7	rst 48	set 6,a	
248	**SAVE**	F8	ret m	set 7,b	
249	**RANDOMIZE**	F9	ld sp,hl	set 7,c	
250	**IF**	FA	jp m,NN	set 7,d	
251	**CLS**	FB	ei	set 7,e	
252	**DRAW**	FC	call m,NN	set 7,h	
253	**CLEAR**	FD	gaat vooraf aan instrukties met iy	set 7,l	
254	**RETURN**	FE	cp N	set 7,(hl)	
255	**COPY**	FF	rst 56	set 7,a	

BIJLAGE B

Meldingen

Rapportages verschijnen onderin het beeld telkens als de computer stopt met iets in BASIC uit te voeren. De rapportage geeft aan waarom de computer stopte; ofwel een natuurlijke oorzaak, ofwel omdat er iets fout ging.
De rapportage heeft een codegetal of -letter, zodat u de melding in de volgende tabel kunt opzoeken. Daarnaast heeft elke rapportage een korte omschrijving van wat er gebeurde en het regelnummer en het opdrachtnummer binnen de regel waar de computer stopte. (Een direkte opdracht wordt weergegeven als regel Ø. Binnen een regel is opdracht 1 aan het begin, opdracht 2 komt na de eerste dubbele punt of **THEN** enzovoorts.)
Het gedrag van **CONTINUE** hangt nauw samen met de rapportages. Normaal gesproken gaat **CONTINUE** naar de regel en de opdracht die in de laatste rapportage genoemd werden, maar er zijn uitzonderingen bij de meldingen Ø, 9 en D (zie ook bijlage C).
Hier vindt u een tabel met alle rapportages. De tabel vermeldt tevens onder welke omstandigheden de melding plaats kan vinden. Dit verwijst weer naar bijlage C. Bijvoorbeeld de foutmelding **A Invalid argument** (ongeldig argument) kan voorkomen bij een **SQR, IN, ACS** en **ASN**. Als u een van deze funkties opzoekt in bijlage C ziet u welke argumenten wel toegestaan zijn.

Code	*Betekenis*	*Situatie*
Ø	OK Succesvolle afronding van het programma, of een sprong naar een regelnummer groter dan enig bestaand nummer. Deze melding verandert de regel en de opdracht waarnaar **CONTINUE** springt niet.	Elke
1	NEXT without FOR (NEXT zonder FOR) - De controlevariabele bestaat niet (is niet opgesteld door een **FOR** opdracht), maar er is een gewone variabele met dezelfde naam.	**NEXT**

Code	Betekenis	Situatie
2	Variabele not found (Variabele niet gevonden) - Bij een eenvoudige variabele zal dit gebeuren als de variabele gebruikt wordt voordat deze een waarde toegekend heeft gekregen door een **LET, READ** of **INPUT** opdracht, of geladen is van band of opgezet is door een **FOR** opdracht. Bij een variabele met onderschrift gebeurt het als de variabele gebruikt wordt voordat deze gedimensioneerd is door een **DIM** opdracht, of geladen is van band.	Elke
3	Subscript wrong (Verkeerd onderschrift) - Een onderschrift ligt buiten het bereik van de array, of er is een verkeerd aantal onderschriften. Als het onderschrift negatief of groter dan 65535 is, wordt fout B gerapporteerd.	variabelen met onderschrift, substrings
4	Out of memory (Te weinig geheugen) - Er is niet genoeg ruimte in de computer voor wat u wilt doen. Als de computer echt vast lijkt te zitten in een dergelijk geval, kan het nodig zijn dat u de opdrachtregel moet wissen door middel van **DELETE**. Vervolgens moet u een paar programmaregels verwijderen om wat speelruimte te maken, zodat u bijvoorbeeld **CLEAR** kunt doen.	**LET, INPUT, FOR, DIM, GO SUB, LOAD, MERGE**. Soms tijdens het uitwerken van uitdrukkingen.
5	Out of screen (Buiten het beeld) - Een **INPUT** opdracht heeft geprobeerd meer dan 23 regels in het onderste deel van het scherm te zetten. Komt ook voor bij een **PRINT AT** 22,..	**INPUT, PRINT AT**
6	Number to big (Getal te groot) - Berekeningen hebben geleid tot een getal groter dan ongeveer 10^{38}.	Alle rekenkunde
7	RETURN without GO SUB (RETURN zonder GO SUB) - Er is een **RETURN** meer dan er **GOSUB**s zijn.	**RETURN**

Code	Betekenis	Situatie
8	End of file (Einde van bestand)	Microdrive etc. bewerkingen.
9	STOP statement (STOP-opdracht) - Hierna zal **CONTINUE** de **STOP** niet herhalen, maar verder gaan met de opdracht erachter.	**STOP**
A	Invalid argument (Ongeldig argument) - Het argument bij een funktie klopt niet.	**SQR, LN, ASN, ACS, USR** (met stringargument)
B	Integer out of range (Integer buiten het bereik) - Als er een integer gevraagd wordt (geheel getal), wordt een argument met een vlottende komma afgerond naar de dichtstbijzijnde integer. Als deze integer buiten het toegestane bereik valt, volgt foutmelding B. Voor toegang tot een array, zie ook fout 3.	**RUN, RANDOMIZE, POKE, DIM, GO TO, GO SUB, LIST, LLIST, PAUSE, PLOT, CHR$, PEEK, USR** (met numeriek argument) Array toegang
C	Nonsense in BASIC (Onzin in BASIC) - De tekst van het (string)argument vormt geen geldige uitdrukking.	**VAL, VAL$**
D	BREAK - CONT repeats (BREAK - CONT gaat door) - **BREAK** werd ingedrukt tijdens een taak waarbij de computer een van de randapparaten gebruikt. Het gedrag van **CONTINUE** na deze melding is normaal en herhaalt de opdracht. Vergelijk dit ook met rapportage L.	**LOAD, SAVE, VERIFY, MERGE, LPRINT, LLIST, COPY.** Tevens als de computer **scroll?** vraagt en u **N, SPACE** of **STOP** indrukt
E	Out of DATA (Zonder gegevens) - U heeft geprobeerd vaker te **READ**en dan de **DATA** lijst lang is.	**READ**
F	Invalid file name (Ongeldige bestandsnaam) - **SAVE** met een naam die leeg was of langer dan 10 karakters.	**SAVE**

Code	Betekenis	Situatie
G	No room for line (Geen ruimte voor de regel) - Er is niet genoeg geheugenruimte om de nieuwe programmaregel op te slaan.	Invoeren van een regel
H	STOP in INPUT (STOP tijdens INPUT) - Een **INPUT** gegeven begon met **STOP**, of dit was - bij **INPUT LINE** - ingedrukt. In tegenstelling tot melding 9, gaat **CONTINUE** na melding H normaal door, en herhaalt de **INPUT** opdracht.	**INPUT**
I	FOR without NEXT (FOR zonder NEXT) - Er was een **FOR** lus die nul keer uitgevoerd moest worden (bijvoorbeeld **FOR** n=1 **TO** 0) en de corresponderende **NEXT** opdracht kon niet gevonden worden.	**FOR**
J	Invalid I/O device (Ongeldig I/O toestel)	Microdrive etc. bewerkingen
K	Invalid colour (Ongeldige kleur)-Het opgegeven getal is geen kleurcode	**INK, PAPER, BORDER, FLASH, BRIGHT, INVERSE, OVER**; tevens na een van de overeenkomende controlekarakters
L	BREAK into program (BREAK in het programma) - **BREAK** ingedrukt. Dit wordt gemerkt tussen twee opdrachten in. De regel en het opdrachtnummer in de rapportage slaan op de opdracht vlak voor de **BREAK** ingedrukt werd. **CONTINUE** springt echter naar de opdracht erachter (sprongen toegestaan), dus wordt er geen opdracht herhaald.	Elke
M	RAMTOP no good (Verkeerde RAMTOP) - Het opgegeven getal voor de RAMTOP is te groot of te klein.	**CLEAR**; soms in **RUN**

Code	Betekenis	Situatie
N	Statement lost (Opdracht verloren) - Sprong naar een niet langer bestaande opdracht of funktie.	**RETURN, NEXT, CONTINUE**
O	Invalid stream (Ongeldige stroom)	Microdrive etc. bewerkingen
P	FN without DEF (FN zonder DEF) - Gebruiker-definieerbare funktie	**FN**
Q	Parameter error (Argumenten fout) - Verkeerde aantal argumenten, of een van de argumenten is van het verkeerde type (string in plaats van getal of vice versa).	**FN**
R	Tape loading error (Fout bij het inlezen van cassette) - Een bestand werd op de cassette gevonden, maar kon niet ingelezen worden, of kon niet gecontroleerd worden.	**VERIFY, LOAD** of **MERGE**

BIJLAGE C(1)

Een beschrijving van de ZX Spectrum om na te slaan

Het eerste deel van deze bijlage is een herhaling van het deel uit de introduktie die het toetsenbord en het scherm behandelt.

Het toetsenbord
SPECTRUM karakters bevatten niet alleen de enkele *symbolen* (letters, cijfers, etc.), maar tevens de *tokens*=tekens (sleutelwoorden, funktienamen, etc.) en deze worden alle vanaf het toetsenbord ingevoerd in plaats van geheel uitgespeld te worden. Om al deze funkties en opdrachten te kunnen bevatten hebben sommige toetsen vijf of meer verschillende betekenissen. De verschillende betekenissen krijgt u deels door het gebruiken van de verschillende SHIFT toetsen (bijvoorbeeld het ingedrukt houden van de **CAPS SHIFT** toets of de **SYMBOL SHIFT** toets tijdens het intikken van de gewenste toets) en deels door de computer in verschillende *modi* (= modussen) te laten zijn.
De modus wordt aangegeven door de *cursor*, een knipperende letter die toont waar het volgende karakter van het toetsenbord geplaatst zal worden.
K (van Keywords = sleutelwoord) modus vervangt automatisch de L modus als de machine een commando of een programmaregel verwacht (in plaats van **INPUT** gegevens), en door de plaats op de regel waar de cursor zich op dat moment bevindt weet de computer of het een regelnummer of een sleutelwoord moet verwachten. Dit gebeurt altijd aan het begin van een regel, net achter **THEN** of net achter **:** (behalve in een string). Als er geen SHIFT toetsen worden gebruikt wordt de eerstvolgende toets geïnterpreteerd als een keyword (op de toetsen) of een cijfer.
L (van Letters) modus is de modus die normaal gesproken, in alle andere gevallen dan hierboven genoemd, in werking is. Als er geen SHIFT toetsen worden gebruikt, wordt het eerstvolgende karakter geïnterpreteerd als het hoofdsymbool op de toets, bij letters de kleine letter.
In zowel K als L modus wordt het gebruik van **SYMBOL SHIFT** en een toets geïnterpreteerd als het kleine rode karakter op de toets en wordt **CAPS SHIFT** met een cijfer geïnterpreteerd als de controlefunktie die in wit boven de toets staat. **CAPS SHIFT** met een van de andere toetsen beïnvloedt de sleutelwoorden niet (in K modus). In L modus worden de letters als hoofdletters weergegeven.
C (van Capitals = hoofdletters) modus is een variant van de L modus, waarin alle letters als hoofdletters weergegeven worden. **CAPS LOCK** zorgt voor een verandering van L modus naar C modus of andersom.

E (van Extended = uitgebreide) modus wordt gebruikt om de resterende karakters, meestal tekens, te verkrijgen. Deze modus treedt in werking door allebei de SHIFT toetsen tegelijkertijd in te drukken, en blijft slechts een toetsindruk in werking. In deze modus geeft een lettertoets een karakter of teken (groen boven de toets) bij gebruik zonder SHIFT. Bij gebruik met SHIFT worden het karakter of teken dat in rood onder de toets staat weergegeven. Een cijfertoets geeft een teken als tegelijkertijd de **SYMBOL SHIFT** wordt gebruikt. Anders krijgt u een kleurcontrolekarakter.

G (van Graphics = grafiek) modus gaat werken nadat **GRAPHICS** (**CAPS SHIFT** en **9**) ingedrukt is en duurt voort tot dit of **9** opnieuw ingedrukt wordt. Een cijfer geeft een mozaïek grafisch symbool, sla **GRAPHICS** en **DELETE** over, en elke letter behalve V, W, X, Y en Z geven een door de gebruiker gedefinieerd karakter.

Elke toets die langer dan twee of drie seconden ingedrukt wordt, zal zichzelf gaan herhalen.

Invoer vanaf het toetsenbord verschijnt onderaan het scherm, precies zoals het wordt ingetikt. Elk karakter (een enkel symbool of een teken) wordt net voor de cursor ingevoegd. De cursor wordt naar links verplaatst met **CAPS SHIFT** en **5** en naar rechts met **CAPS SHIFT** en **8**. Het laatste karakter voor de cursor kan verwijderd worden met **DELETE** (**CAPS SHIFT** en **0**). (Opmerking: de gehele regel wordt verwijderd door **EDIT** (**CAPS SHIFT** en **1**) en daarna **ENTER**).

Zodra **ENTER** wordt ingedrukt wordt de regel uitgevoerd, in het programma opgeslagen of als **INPUT** gegeven gebruikt naar gelang op dat moment de bedoeling is. Dit gebeurt niet als in de regel een 'syntax error' (vergissing in de grammatika) voorkomt. In dat geval wordt een knipperende **?** vlak naast de vergissing neergezet.

Terwijl programmaregels ingevoerd worden, wordt een listing (weergave van programmaregels) bovenaan het beeld gegeven. De meest recente regel die ingevoerd werd, wordt de *current* (tegenwoordige) regel genoemd en wordt aangegeven door het symbool **>**. Dit symbool kan verplaatst worden door gebruik te maken van de toetsen ↓ (**CAPS SHIFT** en **6**) en ↑ (**CAPS SHIFT** en **7**). Als **EDIT** (**CAPS SHIFT** en **1**) ingedrukt wordt, wordt de tegenwoordige regel naar de onderkant van het scherm gebracht en kan daar verbeterd worden.

Als een opdracht uitgevoerd is, of een programma doorlopen, wordt de uitvoer op de bovenste helft van het beeldscherm weergegeven. Dit blijft hier staan totdat **ENTER** getikt wordt met een blanko regel, of ↑ of ↓ wordt ingedrukt. Helemaal beneden aan verschijnt een rapportage door middel van een code (cijfer of letter) die slaat op een mededeling uit bijlage B. Deze rapportage stelt tevens de K modus voor en blijft op het scherm totdat er een toets ingedrukt wordt.

Onder enkele omstandigheden werkt de **CAPS SHIFT** met de **SPACE** toets als een **BREAK** die ervoor zorgt dat de computer stopt en een melding **D** of **L** geeft. Dit gebeurt bij:

a) het einde van een statement terwijl het programma doorlopen wordt, of
b) terwijl de computer de cassetterecorder of de printer gebruikt.

Het televisiescherm

Het scherm heeft 24 regels van elk 32 karakters lang, en wordt verdeeld in twee delen. Het bovenste deel heeft maximaal 22 regels en geeft ofwel een programmalisting ofwel de uitvoer van een programma weer. Als bij het weergeven de onderste regel bereikt is, rolt het hele beeld een regel naar boven. Als dit zou betekenen dat u een regel verliest zonder een redelijke tijd om deze te bekijken, stopt de computer en vraagt **scroll?**. Door de **N**, de **SPACE** of de **STOP** in te drukken stopt het programma met het weergeven van de rapportage **D BREAK - CONT repeats**. Elke andere toets zorgt ervoor dat het rollen (scrollen) van het beeld doorgaat. Het onderste deel van het scherm wordt gebruikt voor het invoeren van opdrachten, programmaregels en **INPUT** gegevens, en voor het weergeven van rapportages. Het onderste deel begint met twee regels, maar breidt zich net zo veel naar boven uit als nodig is voor de hierboven genoemde zaken. Als de onderste helft tekst op de bovenste helft ontmoet, schuift de laatste een regel naar boven op.

Elke karakterpositie heeft *kenmerken* die de *papier* (achtergrond)- en de *inkt* (voorgrond)-kleuren, de helderheid en het al dan niet knipperen bepalen. De mogelijke kleuren zijn zwart, blauw, rood, paars, groen, geel en wit.

De rand van het scherm kan elk van deze kleuren aannemen door de border (rand)-opdracht.

Een karakterpositie is verdeeld in 8 x 8 lichtpunten (pixels) en grafiek met hoge resolutie kan verkregen worden door de pixels individueel aan te sturen (bepalen van inkt- en achtergrondkleur).

De kenmerken van een karakterpositie worden elke keer als er op die positie geschreven wordt, of een pixel ingekleurd wordt, bijgesteld. De precieze manier van deze aanpassing is afhankelijk van de *afdruk parameters*, waarvan twee sets bestaan (*permanente* en *tijdelijke*), elk zes: de **PAPER, INK, FLASH, BRIGHT, INVERSE** en **OVER** parameters. De permanente parameters voor het bovenste deel van het scherm worden vastgesteld door **PAPER, INK** etc. opdrachten, en blijven gelden totdat ze herroepen worden. (Zonder opdracht zijn ze in het begin; zwarte inkt op wit papier, normale helderheid, geen geknipper, normale video en geen transparant afdrukken.) De permanente parameters voor het onderste deel van het scherm gebruiken de randkleur als papier kleur, zwarte of witte inkt om te contrasteren, normale helderheid, geen geknipper, normale video, en geen transparant afdrukken. De tijdelijke parameters worden ingesteld door **PAPER, INK** etc. items, die voorkomen binnen **PRINT, LPRINT, INPUT, PLOT, DRAW** en **CIRCLE** opdrachten, en door **PAPER, INK** etc. controlekarakters als deze naar het televisiescherm gestuurd worden. Ze worden gevolgd door een byte die de waarde van de parameter aangeeft. Tijdelijke parameters gelden slechts tot het einde van de **PRINT** of andere opdracht. In het geval van een **INPUT** opdracht gelden ze tot **INPUT** gegevens nodig zijn van het toetsenbord, dan worden ze vervangen door permanente parameters.

PAPER en **INK** parameters liggen in het bereik van 0 tot 9. De parameters 0 tot 7 geven de kleur waarin het karakter afgedrukt wordt weer:

0	zwart	4	groen
1	blauw	5	cyaan (lichtblauw)
2	rood	6	geel
3	paars	7	wit

Parameter 8 ('transparant') geeft aan dat de kleur op het scherm onveranderd moet blijven wanneer er een karakter gedrukt wordt.
Parameter 9 ('contrast') geeft aan dat de aangegeven kleur (papier of inkt) wit of zwart gemaakt moet worden om te contrasteren met de andere kleur.
De parameters voor **FLASH** en **BRIGHT** zijn 0, 1 of 8: 1 betekent dat het knipperen of de extra helderheid aan is, 0 betekent uit en 8 ('transparant') betekent dat de aangegeven effecten nveranderd blijven.
OVER en **INVERSE** hebben als parameters 0 of 1.

OVER 0	nieuwe karakters wissen de oude karakters
OVER 1	de bitpatronen van de oude en de nieuwe karakters worden gecombineerd door een '*exclusief of*' (exclusive or) bewerking. Dit heet *overdrukken*.
INVERSE 0	nieuwe karakters worden afgedrukt met inktkleur op papierkleur (*normale video*)
INVERSE 1	nieuwe karakters worden afgedrukt met papierkleur op inktkleur (*inverse video*)

Als de televisie een **TAB** controlekarakter ontvangt, worden er nog twee bytes verwacht die een **TAB**-stop n aangeven (minst betekenende byte eerst). Dit wordt eerst verminderd modulo 32, en vervolgens worden zoveel spaties afgedrukt, dat de printpositie terechtkomt in de door de rest van de deling aangegeven kolom.
Als er een komma-controlekarakter ontvangen wordt, worden er voldoende spaties afgedrukt (minimaal 1) om de printpositie te verschuiven naar kolom 0 of kolom 16.
Als er een **ENTER** controlekarakter komt, wordt de printpositie verschoven naar de volgende regel.

De Printer
Uitvoer naar de ZX printer gaat via een buffer die een regel (32 karakters) lang is. Een regel wordt verzonden indien

(i) de tekst van de ene regel overstroomt in de volgende regel,
(ii) een **ENTER** controlekarakter ontvangen wordt,
(iii) bij het einde van het programma nog onafgedrukte tekst aanwezig is,
(iv) een **TAB** controle of een komma-controle de printpositie naar een nieuwe regel stuurt.

TAB en komma-controlekarakters zorgen op dezelfde wijze als op de televisie voor spaties.
De **AT** opdracht verandert de printpositie door het kolomnummer te veranderen. Het regelnummer wordt echter verwaarloosd.
De printer wordt door **INVERSE** en **OVER** op dezelfde wijze beïnvloedt als het televisiescherm. **PAPER, INK, FLASH** en **BRIGHT** werken niet bij de printer.
De printer stopt met de melding B als **BREAK** ingedrukt wordt.
Als de printer niet aangesloten is, wordt de uitvoer gewoon geloosd.

BIJLAGE C(2)

BASIC

Getallen worden tot een nauwkeurigheid van 9 of 1Ø decimalen opgeslagen. Het grootste getal dat u krijgen kunt is ongeveer 10^{38}, en het kleinste (positieve) getal is ongeveer $4 * 10^{-39}$.
Een getal wordt in de ZX Spectrum opgeslagen als een vlottende punt binair getal met een exponent byte e (1 < = e < =255), en vier mantisse bytes m (1/2 < =m < 1). Dit staat voor het getal $m*2^{-128}$.
Omdat 1/2 < =m < 1, is de meest betekenende bit (most significant bit) van de mantisse m altijd 1. Daarom kunnen we deze vervangen door een bit dat het teken weergeeft. Ø voor positieve getallen, 1 voor negatieve.
Kleine integers hebben een speciale representatie waarin de eerste byte Ø is, de tweede byte een tekenbyte is (Ø of FFh) en de derde en vierde de integer vormen in twee-complement vorm, de minst betekenende byte eerst.
Numerieke variabelen hebben namen met een onbepaalde lengte. De naam moet met een letter beginnen, en mag alleen letters en cijfers bevatten. Spaties en kleurencodes worden genegeerd en alle letters worden omgezet in kleine letters.
Controlevariabelen van **FOR..NEXT** lussen kunnen alleen een enkele letter als naam hebben.
Numerieke arrays hebben namen van een enkele letter. Deze kan hetzelfde zijn als de naam van een normale variabele. U kunt net zoveel dimensies maken als u zelf wilt, de lengte van een array is onbeperkt. De onderschriften beginnen bij 1.
Strings zijn flexibel in lengte. De naam van een string is een enkele letter gevolgd door een $.
Stringarrays kunnen ook net zoveel en lang gemaakt worden als u zelf wilt. De naam moet een enkele letter zijn gevolgd door een $. Deze naam mag niet gelijk zijn aan de naam van een string. Alle strings in een bepaalde array hebben dezelfde vastgelegde lengte, die opgegeven wordt door een extra dimensie aan het eind van de **DIM** opdracht. De onderschriften beginnen met 1.
Splitsen: Substrings van strings kunnen aangegeven worden door het gebruik van *splitsers*. Een splitser kan zijn:

(i) leeg
of
(ii) een numerieke uitdrukking
of
(iii) een facultatieve numerieke uitdrukking **TO** een facultatieve numerieke uitdrukking

en wordt gebruikt om een substring weer te geven door

(a) stringuitdrukking (splitser)
(b) stringarrayvariabele (onderschrift, ..., onderschrift, splitser)

hetgeen hetzelfde betekent als

stringarrayvariabele (onderschrift, ..., onderschrift)(splitser)

In het geval (a); stel dat de stringuitdrukking de waarde s$ heeft.
Als de splitser leeg is, is het resultaat s$, hetgeen dan als een substring van zichzelf wordt beschouwd.
Als de splitser een numerieke uitdrukking is met m als waarde, is het resultaat het m-de karakter van s$ (een substring met lengte 1).
Als de splitser vorm (iii) heeft, stel dan dat de eerste numerieke uitdrukking de waarde m heeft (standaard ingestelde waarde is 1), en de tweede heeft de waarde n (standaard ingestelde waarde is de lengte van s$).
Als 1 < =m < =n < =lengte van s$, dan is het resultaat de substring van s$ die begint met het m-de karakter en eindigend met het n-de karakter.
Als 0 < =n < m, dan is het resultaat een lege string.
In andere gevallen treedt foutmelding 3 op.
Het splitsen vindt plaats voor de funkties of bewerkingen worden uitgevoerd, tenzij haakjes iets anders opdragen.
Aan substrings kunnen waarden toegekend worden (zie **LET**).
Als aanhalingstekens binnen een string geschreven moeten worden, moeten deze dubbel weergegeven worden.

Funkties
Het argument van een funktie heeft geen haakjes nodig als het een constante of een (eventueel onderschreven of gesplitste) variabele is.

Funktie	*Soort argument (x)*	*Resultaat*
ABS	getal	Absolute grootheid
ACS	getal	Arccosinus in radialen Fout A als x niet tussen -1 en +1 ligt.
AND	binaire bewerking, de rechter operand is altijd een getal. Numerieke linker operand: String linker operand:	A **AND** B = A als B < > 0 0 als B = 0 A$ **AND** B = A$ als B < > 0 " " als B = 0
ASN	getal	Arcsinus in radialen. Fout A als x niet tussen -1 en +1 ligt.
ATN	getal	Arctangens in radialen

Funktie	*Soort argument (x)*	*Resultaat*
ATTR	twee argumenten, x en y, beide argumenten; tussen haakjes	Een getal waarvan de binaire vorm de kenmerken van regel x, kolom y op het televisiescherm codeert. Bit 7 (meest betekenende) is 1 voor knipperend en Ø voor niet knipperend. Bit 6 is 1 voor helder en Ø voor normaal. Bits 5 tot 3 zijn de papier kleur. Bits 2 tot Ø zijn de inkt kleur. Fout B tenzij Ø < =x < =23 en Ø < =y < =31
BIN		Dit is geen echte funktie, maar een andere notatie voor getallen: **BIN** gevolgd door een opeenvolging van Ø-en en 1-en is het getal dat deze representatie in binaire vorm heeft.
CHR$	getal	Het karakter waarvan de code x is, afgerond naar de dichtstbijzijnde integer
CODE	string	De code van het eerste karakter in x (of Ø als x een lege string is)
COS	getal (in radialen)	Cosinus x
EXP	getal	e^x
FN		**FN** gevolgd door een letter roept een gebruiker--definieerbare funktie op (zie **DEF**). De argumenten moeten tussen haakjes staan; zelfs als er geen argumenten zijn, moeten er toch haakjes geplaatst worden.
IN	getal	Het resultaat van het invoeren op processorniveau van poort x (Ø < =x < =FFFFh) (laadt het bc registerpaar met x en doet de assembleertaal instruktie in a(c))
INKEY$	geen	Leest het toetsenbord. Het resultaat is het karakter dat (in **LL** of **CC** modus) de ingedrukte toets weergeeft, als dat er precies een is, anders is het de lege string
INT	getal	Integer deel (altijd naar beneden afgerond)
LEN	string	Lengte
LN	getal	Natuurlijke logarithme (basis e) Fout A als x < =Ø
NOT	getal	Ø als x < >Ø, 1 als x=Ø. **NOT** heeft rangorde 4
OR	binaire bewerking, beide operands zijn getallen	a **OR** b is 1 als b < >Ø, en is a als b=Ø. **OR** heeft rangorde 2

Funktie	*Soort argument (x)*	*Resultaat*
PEEK	getal	De waarde in de geheugenbyte met adres x (afgerond naar de dichtstbijzijnde integer) Fout B als x niet in het bereik van Ø tot 65535 ligt
PI	geen	pi (3.14159265....)
POINT	twee argumenten, x en y. Allebei getallen, tussen haakjes geplaatst	1 als de pixel op (x,y) inktkleur heeft en Ø als het papierkleur heeft. Fout B tenzij Ø < =x < =255 en Ø < =y < =175
RND	geen	Het volgende pseudo-willekeurige getal in een rij die verkregen wordt door de machten van 75 modulo 65537 met 1 te verminderen en te delen door 65536. Ø < =y < 1
SCREEN$	Twee argumenten, x en y. Allebei getallen, tussen haakjes	Het karakter dat op de televisie verschijnt (normaal of inverse) op regel x, kolom y. Geeft een lege string als het karakter niet herkend wordt. Fout B tenzij Ø < =x < -23 en Ø < =y < 31
SGN	getal	Het teken (-1 voor negatief, Ø voor nul of +1 voor positief) van x
SIN	getal (in radialen)	Sinus x
SQR	getal	Vierkants wortel Fout A als x < Ø
STR$	getal	De karakterstring die afgedrukt zou worden als x afgedrukt werd
TAN	getal (in radialen)	Tangens
USR	getal	Roept de machinecodesubroutine aan waarvan het beginadres x is. Na terugkeer is het resultaat de inhoud van het bc registerpaar
USR	string	Het adres van het bitpatroon van het gebruiker-definieerbare karakter aangegeven door x Fout A als x niet een enkele letter tussen a en u, of een gebruiker-definieerbare grafiek is
VAL	string	Werkt x (zonder de aanhalingstekens) uit als een numerieke uitdrukking Fout C als x een grammatikale fout bevat, anders een numerieke uitdrukking. Andere fouten afhankelijk van de inhoud van de string mogelijk

Funktie	*Soort argument (x)*	*Resultaat*
VAL$	string	Werkt x (zonder de aanhalingstekens) uit als een stringuitdrukking Fout C als x een grammatikale fout bevat, anders een numerieke waarde. Andere rapportages mogelijk, zie **VAL**.
-	getal	Ontkenning

+	Optelling (van getallen), samentrekking (van strings)	
-	Aftrekking	
*	Vermenigvuldiging	
/	Deling	
↑	Machtsverheffen. Fout B als de linker operand negatief is	
=	gelijk	Beide operands moeten van hetzelfde type zijn. Het resultaat is 1 als de vergelijking waar is, en Ø als de vergelijking niet-waar is.
>	Groter dan	
<	Kleiner dan	
< =	Kleiner dan of gelijk aan	
> =	Groter dan of gelijk aan	
< >	Niet gelijk aan	

Funkties en bewerkingen hebben de volgende rangorde:

Bewerking	*Rangorde*
Onderschriften en splitsen	12
Alle funkties behalve **NOT** en ontkenning	11
↑	1Ø
Ontkenning (minus om iets negatief te maken)	9
*, /	8
+, - (minus om een getal van een ander af te trekken)	6
=, >, <, < =, > =, < >	5
NOT	4
AND	3
OR	2

Opdrachten

In deze lijst zijn de volgende symbolen gebruikt

a	een enkele letter
v	een variabele
x, y, z	numerieke uitdrukkingen
m, n	numerieke uitdrukkingen afgerond naar de dichtstbijzijnde integer
e	een uitdrukking
f	een uitdrukking met stringwaarde
s	een opeenvolging van opdrachten gescheiden door komma's
c	een opeenvolging van kleuritems, elk afgesloten door komma's, of puntkomma's. Een kleuritem heeft als vorm: **PAPER, INK, FLASH, BRIGHT, INVERSE** of **OVER** opdracht.

Merk op dat elke uitdrukking overal gebruikt mag worden (behalve het regelnummer aan het begin van een opdracht).

Alle opdrachten behalve **INPUT, DEF** en **DATA** kunnen als direkte opdracht of in een programma gebruikt worden (hoewel soms de ene manier van gebruik logischer is dan de andere). Een direkte opdracht, of een programmaregel kan uit verschillende opdrachten bestaan, gescheiden door dubbele punten (:). Er is geen vastgestelde volgorde van opdrachten. Alleen bij de **IF** en de **REM** opdrachten moet u even opletten.

BEEP x,y	Laat een toon via de luidspreker klinken. x seconden lang en y halve tonen boven de midden-C (of eronder als het een negatieve waarde is)
BORDER m	Stelt de kleur van de rand van het scherm in, en tevens de kleur van het papier van het onderste deel van het scherm. Fout K als m niet tussen Ø en 7 ligt.
BRIGHT n	Stel helderheid van nog af te drukken karakters in. n=Ø voor normaal, 1 voor helder, 8 voor transparant Fout K als n niet Ø, 1 of 8 is.
CAT	Werkt niet zonder Microdrive, etc.
CIRCLE x,y,z	Tekent een cirkelboog, middelpunt (x,y), straal z
CLEAR	Wist alle variabelen, waarbij geheugenruimte vrij komt. Tevens vinden **RESTORE** en **CLS** plaats, wordt de **PLOT** positie ingesteld op de linkerbenedenhoek, en wist de **GO SUB** stapel

CLEAR n — Als **CLEAR**, maar verandert, indien mogelijk, de variabele **RAMTOP** van het systeem in n, en plaats de nieuwe **GO SUB** stapel daar

CLOSE# — Werkt niet zonder Microdrive, etc.

CLS — Wist het beeldscherm

CONTINUE — Continueert het programma, beginnend waar de computer gebleven was toen er gestopt werd, zolang de rapportage niet 0 was. Als de rapportage 9 of L was, wordt er vervolgd met de eerstvolgende opdracht (er wordt gelet op eventuele sprongen); anders wordt de opdracht waar de fout gemeld werd herhaald.
Als de laatste rapportage in een direkte opdrachtregel plaatsvond zal **CONTINUE** proberen verder te gaan in de regel en in een lus gaan als de foutmelding kwam in 0*1, een rapportage geven als de foutmelding kwam in 0:2, of een fout N geven als de melding in 0:3 of hoger kwam.
CONTINUE staat als **CONT** op het toetsenbord

COPY — Zendt een kopie van de eerste 22 regels van het beeldscherm naar de printer als deze aangesloten is. Als deze niet aangesloten is, gebeurt er niets. Merk op dat **COPY** niet gebruikt kan worden om de automatische listing af te drukken.
Fout D als **BREAK** ingedrukt is

DATA e*1*, e*2*, e*3*, .. — Deel van een **DATA** lijst. Moet in een programma staan

DEF FN *a(a1,...,ak)*=e — Gebruiker-definieerbare funktie definitie; moet in een programma staan. Elke *a*, *a1* tot *ak* moet ofwel een enkele letter of een enkele letter gevolgd door $ (string) zijn
Heeft de vorm **DEF FN** *a*()=e als er geen argumenten zijn

DELETE f — Werkt niet zonder Microdrive, etc.

DIM *a(n1,...nk)* — Wist elke array met als naam *a*, en stelt een nieuwe numerieke array *a* in met k dimensies *n1,...nk*.
Alle waarden worden ingesteld op 0

DIM *a*$(*n1*,...,*nk*) — Wist elke array of string met als naam *a*$, en stelt een nieuwe karakterarray in met *k* dimensies *n1*,..*nk*.
Alle waarden worden ingesteld op "". Deze array kan voorgesteld worden als een array met strings met vaste lengte *nk*, en met *k-1* dimensies *n1*,..*nk-1*.
Fout 4 als er geen ruimte voor de array is.
Een array is niet gedefinieerd zolang er niet gedimensioneerd is door een **DIM** opdracht

DRAW x,y — Teken lijn naar x,y (**DRAW** x,y,Ø)

DRAW x,y,z — Tekent een lijn van de huidige **PLOT** positie door x horizontaal te bewegen en y verticaal, daarbij over de hoek z draaiend.
Fout B als de tekening buiten het beeld valt

ERASE — Werkt niet zonder Microdrive, etc.

FLASH — Bepaalt of karakters knipperen of niet. n=Ø voor stabiel, n=1 voor geknipper, n=8 voor geen verandering

FOR *a*=x **TO** y — **FOR** *a*=x **TO** y **STEP** 1

FOR *a*=x **TO** y **STEP** z — Wist elke gewone variabele *a* en stelt in: een controlevariabele *a* met waarde x, een limiet y, een stapgrootte z, en een lusadres dat verwijst naar de opdracht achter de **FOR** opdracht. Controleert of de beginwaarde groter is (als de stapgrootte > =Ø) of kleiner is (als stapgrootte < Ø) dan de limiet, en springt, als dat zo is, naar de opdracht **NEXT** *a*. Daarbij een fout 1 gevend als deze opdracht ontbreekt. Zie ook **NEXT**.
Fout 4 als er geen ruimte voor de controlevariabele is.

FORMAT f — Werkt niet zonder Microdrive,etc

GOSUB n — Stopt het regelnummer van de **GOSUB** opdracht op een stapel; vervolgens als **GO TO** n.
Fout 4 als er niet genoeg **RETURN**s zijn

GO TO n — Springt naar regel n (of als die er niet is, naar de eerste regel erna)

IF x **THEN** s — Als x waar is (niet nul) wordt s uitgevoerd. s staat voor alle opdrachten tot het einde van de regel.
De vorm **IF** x **THEN** regelnummer is niet toegelaten

INK n

Stelt de inkt (voorgrond)-kleur van nog af te drukken karakters in. n ligt tussen Ø en 7 voor een kleur, n=8 voor transparant of 9 voor contrast. Zie ook *het beeldscherm* in bijlage B.
Fout K als n niet ligt tussen Ø en 9

INPUT ...

De ... is een opeenvolging van **INPUT** items, gescheiden door komma's, puntkomma's of aanhalingstekens als in een **PRINT** opdracht. Een **INPUT** item kan bestaan uit:
(i) Elk **PRINT** item dat niet met een letter begint
(ii) Een variabele naam of
(iii) **LINE** en dan een stringvariabele naam.
De **PRINT** items en delers in (i) worden precies zo behandeld als in een **PRINT** opdracht, maar nu wordt alles afgedrukt in het onderste deel van het scherm.
Voor (ii) stopt de computer en wacht op invoer van een uitdrukking van het toetsenbord; de waarde hiervan wordt toegekend aan de variabele. De invoer heeft op de normale manier een echo, en grammatikafouten geven de knipperende **?**. Bij stringuitdrukkingen is de input-buffer ingesteld om twee string aanhalingstekens te bevatten (die kunnen gewist worden indien nodig). Als het eerste karakter van de invoer **STOP** is, stopt het programma met de fout H. (iii) is als (ii) behalve dat de invoer behandeld wordt als een string zonder aanhalingstekens, en de **STOP** opdracht niet werkt. Om te stoppen moet u ↓ gebruiken.

INVERSE n

Bestuurt de omkering (inverse) van de nog af te drukken karakters. Als n=Ø worden de karakters in *normale video*, dat wil zeggen inktkleur op papierkleur, afgedrukt.
Als n=1 worden de karakters in *inverse video*, dat wil zeggen papierkleur op inktkleur, afgedrukt. Zie *Het beeldscherm* bijlage B.
Fout K als n niet Ø of 1 is

LET v=e

Kent de waarde e toe aan de variabele v. **LET** kan niet weggelaten worden. Een normale variabele is ongedefinieerd totdat er een waarde aan is toegekend door een **LET, READ** of **INPUT** opdracht. Als v een stringvariabele met onderschrift is, of een gesplitste stringvariabele (substring), dan is de toekenning volgens de *Procrusteaanse* methode (vaste lengte): De stringwaarde van e

	wordt ofwel ingekort, ofwel aangevuld met spaties aan de rechterkant om hem even lang te maken als de variabele v
LIST	**LIST** 0
LIST n	Toont het programma op het scherm (list). Beginnend bij de eerste regel waarvan het nummer minstens n is. n wordt de current (tegenwoordige) regel.
LLIST	**LLIST** 0
LLIST n	Als **LIST** maar met gebruik van een printer
LOAD f	Laadt programma en variabelen
LOAD f **DATA** ()	Laadt een numerieke array
LOAD f **DATA** $()	Laadt een karakterarray $
LOAD f **CODE** m,n	Laadt maximaal n bytes, beginnend bij adres m
LOAD f **CODE** m	Laadt bytes beginnend bij adres m
LOAD f **CODE**	Laadt bytes terug naar het adres van waar ze gesaved werden
LOAD f **SCREEN$**	**LOAD** f **CODE** 16384,6912 Zoekt naar een bestand van de juiste soort op een cassettebandje en laadt dit, daarbij oudere versies in het geheugen wissend. Zie hoofdstuk 20
LPRINT	Als **PRINT** maar met gebruik van een printer
MERGE f	Als **LOAD** f, maar oude programmaregels en variabelen worden niet gewist, tenzij er nieuwe zijn met hetzelfde nummer of dezelfde naam
MOVE *f1,f2*	Werkt niet zonder Microdrive, etc.
NEW	Start het BASIC-systeem opnieuw, daarbij programma's en variabelen wissend. Het geheugen tot en met het adres aangegeven door RAMTOP wordt in beslag genomen. De systeemvariabelen UDG, P RAMT, RASP en PIP worden ingesteld

NEXT *a* — (i) vindt de controlevariabele *a*
(ii) telt de stapgrootte op
(iii) als de stapgrootte > = 0 en de waarde > de limiet, of de stapgrootte < 0 en de waarde < limiet, dan wordt er naar de lus opdracht gesprongen.
Fout 2 als er geen variabele *a* is
Fout 1 als er wel een controlevariabele is, maar deze niet *a* luidt

OPEN # — Werkt niet zonder Microdrive, etc.

OUT m,n — Voert byte n bij poort m op processorniveau uit. (Laadt het bc registerpaar met m, het a register met n, en voert de assembleertaal routine out (c),a uit.)
0 < =m < =65535, - 255 < =n < =255, anders fout B

OVER n — Bestuurt het overdrukken van nog af te drukken karakters.
Als n=0 wissen de nieuwe karakters de oude uit.
Als n=1 dan worden de nieuwe karakters gemengd met de oude karakters, zodat er inktkleur is waar een van twee (maar niet allebei) inktkleur had, en papierkleur waar beide papierkleur of beide inktkleur hadden. Zie *Het beeldscherm* - Bijlage B
fout K als n niet 0 of 1 is

PAPER n — Als **INK**, maar besturing van de papier (achtergrond) kleur

PAUSE n — Stopt de uitvoer van het programma en laat het beeld zien gedurende n beelden (bij 50 beelden per seconden of 60 beelden per seconde in USA) of totdat er een toets ingedrukt wordt.
0 < =n < =65535, anders fout B. Als n=0 blijft de pauze duren totdat er een toets ingedrukt wordt

PLOT c;m,n — Tekent een inktpunt (onderworpen aan **OVER** en **INVERSE**) op pixel (m,n); verplaatst de **PLOT** positie
Tenzij de kleuritems c anders beslissen, wordt de inktkleur van de karakterpositie waar de pixel in ligt, veranderd in de huidige permanente inktkleur. De anderen blijven onveranderd.
0 < =m < =255, 0 < =n < =175 anders fout B

POKE m,n — Schrijft de waarde n in de geheugenbyte met adres m
Ø < =m < =65535, - 255 < =n < =255 anders fout B

PRINT... — De ... is een opeenvolging van **PRINT** items, gescheiden door komma's(,), puntkomma's(;), of aanhalingstekens("). Deze worden opgeslagen in het beeldbestand om naar het beeldscherm gestuurd te worden.
Een puntkomma (;) tussen twee items heeft geen effect. Dit wordt alleen gebruikt om de items te scheiden. Een komma voert een komma-controlekarakter uit, een aanhalingsteken voert een **ENTER** uit.
Aan het einde van een **PRINT** opdracht wordt een **ENTER** karakter uitgevoerd, tenzij de **PRINT** opdracht eindigt met een puntkomma, een komma of een aanhalingsteken.
Een **PRINT** item kan zijn;
(i) leeg, dat wil zeggen niets
(ii) een numerieke uitdrukking. Eerst wordt een minteken afgedrukt als de waarde negatief is. Als x de modulus van de waarde is, en $x < = 1Ø^{-5}$ of $x > = 1Ø^{13}$, dan wordt er afgedrukt in de wetenschappelijke notatie. De mantisse heeft maximaal 8 cijfers (geen staartnullen), en de decimale punt (afwezig bij een enkel cijfer getal) staat achter het eerste cijfer. De exponent is E gevolgd door + of -, gevolgd door een of twee cijfers.
In andere gevallen wordt x in normale decimale notatie afgedrukt met maximaal 8 significate decimalen, en geen staart nullen achter de decimale punt. Een decimale punt direkt aan het begin wordt altijd gevolgd door een nul. .Ø3 en Ø.3 worden dus afgedrukt als aangegeven. Ø wordt als een enkel getal Ø afgedrukt.
(iii) een stringuitdrukking
De tekens in de string worden opgeblazen, zo mogelijk met een spatie ervoor en erachter.
Controlekarakters hebben hun controle effect.
Niet-herkenbare karakters worden afgedrukt als een **?**.
(iv) **AT** m,n
Voert een **AT** controlekarakter uit, gevolgd door een byte voor m (het regelnummer) en een byte voor n (het kolomnummer).
(v) **TAB** n
Voert een **TAB** controlekarakter uit gevolgd door twee bytes voor n (minst betekenende byte eerst), de **TAB** stop.

(vi) Een kleuritem, dat de vorm kan krijgen van een **PAPER, INK, FLASH, BRIGHT, INVERSE,** of **OVER** opdracht.

RANDOMIZE — **RANDOMIZE** Ø

RANDOMIZE n — Stelt de systeemvariabele SEED, die de volgende waarde van **RND** genereerd, in. Als n <> Ø, krijgt SEED de opgegeven waarde n; als n=Ø krijgt SEED de waarde van een andere systeemvariabele (FRAMES) die het aantal beelden dat tot nu toe op het beeldscherm getoond is, telt. Hierdoor wordt een benadering van geheel willekeurig verkregen

RANDOMIZE staat als **RAND** op het toetsenbord

Fout B als n niet ligt tussen Ø en 65535

READ *v1, v2, ..., vk* — Kent waarden toe aan de variabelen waarbij de waarden uit een **DATA** lijst gehaald worden.

Fout C als een uitdrukking van het verkeerde type.

Fout E als er nog variabelen over zijn als de **DATA** lijst leeg is

REM ... — Geen effect. '...' kan elke opeenvolging van karakters zijn behalve **ENTER**. Een : is ook toegestaan. Achter een **REM** opdracht worden alle opdrachten en funkties van dezelfde programmaregel genegeerd.

RESTORE — **RESTORE** Ø

RESTORE n — Herstelt de **DATA** wijzer naar de eerste **DATA** opdracht in de regel met minstens nummer n. De volgende **READ** opdracht begint hier met lezen

RETURN — Zoekt een opdracht op de **GO SUB** stapel op, en springt naar de regel daaropvolgend.

Fout 7 als op de **GO SUB** stapel geen opdracht aanwezig is. Er zit ergens in uw programma een vergissing; U heeft waarschijnlijk niet net zoveel **GO SUB**s als **RETURN**s

RUN — **RUN** Ø

RUN n — **CLEAR** en vervolgens **GO TO** n

SAVE f — Bewaart het programma en de variabelen

SAVE f **LINE** m	Bewaart het programma en de variabelen op een speciale manier, zodat als het programma geladen wordt, er onmiddellijk naar regel m gesprongen wordt
SAVE f **DATA** ()	Bewaart een numerieke array
SAVE f **DATA** $()	Bewaart de karakterarray $
SAVE f **CODE** m,n	Bewaart n bytes beginnend bij adres m
SAVE f **SCREEN$**	**SAVE** f **CODE** 16384,6912 Bewaart informatie op cassette onder de naam f. Fout F als f leeg is of een lengte heeft van meer dan 10 karakters. Zie hoofdstuk 20
STOP	Stopt het programma met rapportage 9. **CONTINUE** vervolgt met de volgende opdracht
VERIFY	Hetzelfde als **LOAD**, maar de gegevens worden niet in het RAM geheugen geladen, maar vergeleken met de daar al aanwezige informatie. Fout R als een van de vergelijkingen een verschil in bytes laat zien.

BIJLAGE D

Voorbeeldprogramma's

Deze bijlage bevat een paar voorbeeldprogramma's waarmee de mogelijkheden van de ZX Spectrum gedemonstreerd worden.
Het eerste programma vraagt om een datum als invoer en berekent op welke dag van de week deze datum valt.

```
 10 REM zet datum om naar dag
 20 DIM d$(7,6): REM dagen van de week
 30 FOR n=1 TO 7: READ d$(n): NEXT n
 40 DIM m(12): REM lengte van de maanden
 50 FOR n=1 TO 12: READ m(n): NEXT n
100 REM voer datum in
110 INPUT "dag?";dag
120 INPUT "maand?";maand
130 INPUT "jaar (alleen 20-ste eeuw)?";jaar
140 IF jaar < 1901 THEN PRINT "de 20-ste eeuw begint bij 1901": GO
    TO 100
150 IF jaar > 2000 THEN PRINT "de 20-ste eeuw eindigt in 2000": GO
    TO 100
180 IF jaar/4-INT(jaar/4)=0 THEN LET m(2)=29 : REM schrikkeljaar
190 IF dag > m(maand) THEN PRINT "Deze maand heeft maar ";
    m(maand); " dagen." : GO TO 500
200 IF dag > 0 THEN GO TO 300
210 PRINT "Geen geintjes, een echte datum svp."
220 GO TO 500
300 REM zet datum om in aantal dagen sinds het begin van de eeuw
310 LET y=jaar-1901
320 LET b=365*y+INT(y/4): REM aantal dagen sind begin jaar
330 FOR n=1 TO maand-1 : REM tel vorige maanden erbij op
340 LET b=b+m(n) : NEXT n
350 LET b=b+dag
400 REM zet om naar dag van de week
410 LET b=b-7*INT (b/7)+1
420 PRINT dag; "/" ; maand ; "/" ; jaar
430 FOR n=6 TO 3 STEP -1 : REM verwijder teveel spaties
440 IF d$(b,n)  < > " " THEN GO TO 460
450 NEXT n
```

```
460 LET e$=d$(b, TO n)
470 PRINT "is een "; e$ ; "dag"
500 LET m(2)=28 : REM restore februari
510 INPUT "nogmaals?", a$
520 IF a$="N" THEN GO TO 540
530 IF a$ < > "N" THEN GO TO 100
1000 REM dagen van de week
1010 DATA "Maan", "Dins", "Woens"
1020 DATA "Donder", "Vrij", "Zater", "Zon"
1100 REM lengte van de maanden
1110 DATA 31, 28, 31, 30, 31, 30
1120 DATA 31, 31, 30, 31, 30, 31
```

Het volgende programma is vooral handig als u een bezoek brengt aan het geboorteland van de ZX Spectrum: Engeland. Het programma bewerkt de onderlinge verhoudingen van yards, feet en inches.

```
10 INPUT "yards?", yd, "feet?", ft, "inches?", in
40 GO SUB 2000: REM druk de waarden af
50 PRINT '" = ";
70 GO SUB 1000 : REM de aanpassing
80 GO SUB 2000 : REM druk de nieuwe waarden af
90 PRINT
100 GO TO 10
1000 REM subroutine om yd, ft, in om te zetten naar de normale vorm
     voor yards, feet en inches
1010 LET in=36*yd+12*ft+in : REM nu is alles in inches
1030 LET s=SGN in: LET in=ABS in: REM we werken met in positief,
     het teken wordt opgeslagen in s
1060 LET ft=INT (in/12): LET in=(in-12*ft)*s : REM nu zijn de in goed
1080 LET yd=INT (ft/3)*s: LET ft=ft*s - 3*yd : RETURN
2000 REM subroutine om yd, ft en in af te drukken
2010 PRINT yd; "yd"; ft; "ft"; in; "in";: RETURN
```

Hier een programma voor het werpen van de munten van I Ching (met de patronen ondersteboven).

```
5 RANDOMIZE
10 FOR m=1 TO 6: REM 6 worpen
20 LET c=0 : REM initialiseer munt totaal op 0
30 FOR n=1 TO 3 : REM 3 munten
40 LET c=c+2+INT (2*RND)
50 NEXT n
60 PRINT " ";
```

```
 70 FOR n=1 TO 2: REM 1ste voor het hexagram, 2-de voor de
    veranderingen
 80 PRINT "---";
 90 IF c=7 THEN PRINT "-";
100 IF c=8 THEN PRINT " ";
110 IF c=6 THEN PRINT "X";: LET c=7
120 IF c=9 THEN PRINT "0";: LET c=8
130 PRINT "--- ";
140 NEXT n
150 PRINT
160 INPUT a$
170 NEXT m: NEW
```

Om dit te gebruiken moet u het programma intikken en runnen. Tik vervolgens vijf keer **ENTER** om twee hexagrammen te krijgen. Zoek deze op in het Chinese boek der veranderingen. De tekst zal een situatie beschrijven en de benodigde activiteiten. U zult diep na moeten denken over de parallellen tussen deze tekst en uw eigen leven. Drukt u een zesde keer op **ENTER**, dan vernietigt het programma zichzelf. Dit is bedoeld om u te ontmoedigen om het programma te lichtzinnig te gebruiken.
Veel mensen vinden dat de teksten opmerkelijk toepasselijk zijn voor een kansspel. het is de vraag of dit met uw ZX Spectrum ook opgaat. In het algemeen zijn computers nogal atheïstisch.

Hier vindt u een programma voor het 'miereneter'-spel. U bedenkt een dier, en de computer probeert te raden welk dier u in gedachten heeft, door u allerlei vragen te stellen die u met 'ja' of 'nee' kunt beantwoorden. Als de computer nog nooit van uw beest gehoord heeft, stelt de computer u een vraag de volgende keer gebruik te maken van deze kennis.

```
  5 REM miereneters
 10 LET nq=100 : REM aantal beesten en vragen
 15 DIM q$(nq,50) : DIM a(nq,2) : DIM r$(1)
 20 LET qf=8
 30 FOR n=1 TO qf/2-1
 40 READ q$(n) : READ a(n,1) : READ a(n,2)
 50 NEXT n
 60 FOR n=n TO qf-1
 70 READ q$(n) : NEXT n

100 REM begin het spel
110 PRINT "denk aan een dier.", "Druk op een toets om door te gaan."
120 PAUSE 0
130 LET c=1 : REM begin eerste vraag
```

```
140 IF a(c,1)=0 THEN GO TO 300
150 LET p$=q$(c) : GO SUB 910
160 PRINT "?" : GO SUB 1000
170 LET in=1 : IF r$="j" THEN GO TO 210
180 IF r$="J" THEN GO TO 210
190 LET in=2 : IF r$="n" THEN GO TO 210
200 IF r$ < > "N" THEN GO TO 150
210 LET c=a(c,in) : GO TO 140

300 REM dier
310 PRINT "Denk je aan een"
320 LET p$=q$(c) : GO SUB 900 : PRINT "?"
330 GO SUB 1000
340 IF r$="j" THEN GO TO 400
350 IF r$="J" THEN GO TO 400
360 IF r$="n" THEN GO TO 500
370 IF r$="N" THEN GO TO 500
380 PRINT "Geef netjes antwoord als ik tegen je spreek." : GO TO 300

400 REM geraden
410 PRINT "Dacht ik al." : GO TO 800

500 REM nieuw dier
510 IF gf > nq-1 THEN PRINT "Ik vind jouw dier erg", "interessant,
    maar ik heb", "er geen ruimte meer voor." : GO TO 800
520 LET q$(qf)=q$(c) : REM verplaats oude dier
530 PRINT "Wat is het dan?" : INPUT q$(qf+1)
540 PRINT "Stel me een vraag die ", "onderscheid maakt tussen "
550 LET p$=q$(qf) : GO SUB 900 : PRINT " en"
560 LET p$=q$(qf+1) : GO SUB 900 : PRINT " "
570 INPUT s$ : LET b=LEN s$
580 IF s$(b)="?" THEN LET b=b-1
590 LET q$(c)= s$(TO b) : REM voer vraag in
600 PRINT "Wat is het antwoord op"
610 LET p$=q$(qf+1) : GO SUB 900 : PRINT "?"
620 GO SUB 1000
630 LET in=1 : LET io=2 : REM antwoorden voor nieuwe en oude
    dieren
640 IF r$="j" THEN GO TO 700
650 IF r$="J" THEN GO TO 700
660 LET in=2 : LET io=1
670 IF r$="n" THEN GO TO 700
680 IF r$="N" THEN GO TO 700
690 PRINT " Dat werkt niet. " : GO TO 600
```

```
700 REM antwoorden bijwerken
710 LET a(c,in)=qf+1 : LET a(c,io)=qf
720 LET qf=qf+2 : REM volgende vrije dier-plaats
730 PRINT "Daar had je me te pakken."

800 REM nogmaals?
810 PRINT "Nog een keer?" : GO SUB 1000
820 IF r$="j" THEN GO TO 100
830 IF r$="J" THEN GO TO 100
840 STOP

900 REM druk af zonder extra spaties
905 PRINT " ";
910 FOR n=50 TO 1 STEP -1
920 IF p$(n) < > " " THEN GO TO 940
930 NEXT n
940 PRINT p$(TO n);: RETURN

1000 REM antwoord
1010 INPUT r$ : IF r$="" THEN RETURN
1020 LET r$=r$(1) : RETURN

2000 REM initiele beesten
2010 DATA "Leeft het in de zee",4,2
2020 DATA "Heeft het schubben",3,5
2030 DATA "Eet het mieren",6,7
2040 DATA "een walvis", "een blancmanger", "een miereneter", "een
     mier"
```

Hier is een programma om de Engelse vlag te tekenen.

```
  5 REM Engelse vlag
 10 LET r=2: LET w=7: LET b=1
 20 BORDER 0: PAPER b: INK w: CLS
 30 REM zwarte onderkant van het scherm
 40 INVERSE 1
 50 FOR n=40 TO 0 STEP -8
 60 PLOT PAPER 0;7,n: DRAW PAPER 0; 241,0
 70 NEXT N: INVERSE 0
100 REM teken de witte delen
105 REM St. George
110 FOR n=0 TO 7
120 PLOT 104+n,175: DRAW 0,-35
130 PLOT 151-n,175: DRAW 0,-35
```

```
140 PLOT 151-n,48: DRAW 0,35
150 PLOT 104+n,48: DRAW 0,35
160 NETX n
200 FOR n=0 TO 11
210 PLOT 0,139-n: DRAW 111,0
220 PLOT 255,139-n: DRAW -111,0
230 PLOT 255,84+n: DRAW -111,0
240 PLOT 0,84+n: DRAW 111,0
250 NEXT n
300 REM St. Andrew
310 FOR n=0 TO 35
320 PLOT 1+2*n,175-n: DRAW 32,0
330 PLOT 224-2*n,175-n:DRAW 16,0
340 PLOT 254-2*n,48+n; DRAW -32,0
350 PLOT 17+2*n,48+n:DRAW 16,0
360 NEXT n
370 FOR n=0 TO 19
380 PLot 185+2*n,140+n: DRAW 32,0
390 PLOT 200+2*n,83-n: DRAW 16,0
400 PLOT 39-2*n,83-n: DRAW 32,0
410 PLOT 54-2*n,140+n: DRAW -16,0
420 NEXT n
425 REM vul de extra delen op
430 FOR n=0 TO 15
440 PLOT 255,160+n: DRAW 2*n-30,0
450 PLOT 0,63-n: DRAW 31-2*n,0
460 NEXT n
470 FOR n=0 TO 7
480 PLOT 0,160+n: DRAW 14-2*n,0
485 PLOT 255,63-n: DRAW 2*n-15,0
490 NEXT n
500 REM rode strepen #
510 INVERSE 1
520 REM St. George
530 FOR n=96 TO 120 STEP 8
540 PLOT PAPER r;7,n: DRAW PAPER r;241,0
550 NEXT n
560 FOR n=112 TO 136 STEP 8
570 PLOT PAPER r;n,168: DRAW PAPER R;0,-113
580 NEXT n
600 REM St. Patrick
610 PLOT PAPER r;170,140: DRAW PAPER r;70,35
620 PLOT PAPER r;179,140: DRAW PAPER r;70,35
630 PLOT PAPER r;199,83: DRAW PAPER r;56,-28
```

```
640 PLOT PAPER r;184,83: DRAW PAPER r;70,-35
650 PLOT PAPER r;86,83: DRAW PAPER r;-70,-35
660 PLOT PAPER r;72,83: DRAW PAPER r;-70,-35
670 PLOT PAPER r;56,140: DRAW PAPER r;-56,28
680 PLOT PAPER r;71,140: DRAW PAPER r;-70,35
690 INVERSE 0: PAPER 0: INK 7
```

Als u geen behoefte heeft aan de Engelse vlag, zult u er zelf een moeten tekenen. De Nederlandse vlag is tamelijk gemakkelijk. Voor de Amerikaanse vlag kunt u gebruik maken van een '*'.

Hier een programma om galgje te spelen. De ene speler voert een woord in en de andere probeert het te raden.

```
  5 REM galgje
 10 REM stel het scherm in
 20 INK 0 : PAPER 7: CLS
 30 LET x=240 : GO SUB 1000 : REM teken man
 40 PLOT 238,128 : DRAW 4,0 : REM mond
100 REM stel woord in
110 INPUT w$ : REM te raden woord
120 LET b=LEN w$: LET v$=""
130 FOR n=2 TO b: LET v$=v$+" "
140 NEXT n : REM v$=woord tot nu toe geraden
150 LET c=0 : LET d=0 : REM teller, fouten
160 FOR n=0 TO b-1
170 PRINT AT 2 0 ,n;"-";
180 NEXT n : REM -'s in plaats van letters
200 INPUT "Raad een getal: ";g$
210 IF g$="" THEN GO TO 200
220 LET g$=g$(1) : REM 1-ste letter
230 PRINT AT 0 ,c,g$
240 LET c=c+1: LET u$=v$
250 FOR n=1 TO b : REM stel geraden woord bij
260 IF w$(n)=g$ THEN LET v$(n)=g$
270 NEXT n
280 PRINT AT 19,0 ;v$
290 IF v$=w$ THEN GO TO 500 : REM geraden
300 IF v$ < > u$ THEN GO TO 200 : REM nog niet
400 REM volgende deel van galg
410 IF d=8 THEN GO TO 600 : REM hangen
420 LET d=d+1
430 READ x0 ,y0 ,x,y
440 PLOT x0 ,y0 :DRAW x,y
```

```
 450 GO TO 200
 500 REM vrij
 510 OVER 1 : REM wis man
 520 LET x=240 : GO SUB 1000
 530 PLOT 238,128 : DRAW 4,0 : REM mond
 540 OVER 0 : REM teken man opnieuw
 550 LET x=146: GO SUB 1000
 560 PLOT 143, 129 : DRAW 6,0 ,PI/2 : REM lach
 570 GO TO 800
 600 REM hangen
 610 OVER 1 : REM wis vloer
 620 PLOT 255,65: DRAW -48,0
 630 DRAW 0 ,-48 : REM open valluik
 640 PLOT 238,128 : DRAW 4,0 : REM wis mond
 650 REM beweeg ledematen
 655 REM armen
 660 PLOT 255,117:DRAW -15,-15: DRAW -15,15
 670 OVER 0
 680 PLOT 236,81: DRAW 4,21: DRAW 4,-21
 690 OVER 1 : REM benen
 700 PLOT 255,66: DRAW -15,15: DRAW -15,-5
 710 OVER 0
 720 PLOT 236,6 0 : DRAW 4,21: DRAW 4,-21
 730 PLOT 237,127 : DRAW 6, 0 ,-PI/2 : REM frons
 740 PRINT AT 19, 0 ; w$
 800 INPUT "nogmaals? ";a$
 810 IF a$="" THEN GO TO 850
 820 LET a$=a$(1)
 830 IF a$="n" THEN STOP
 840 IF a$(1)="N" THEN STOP
 850 RESTORE : GO TO 5
1000 REM teken man op kolom x
1010 REM hoofd
1020 CIRCLE x,132,8
1030 PLOT x+4,134: PLOT x-4,134: PLOT x,131
1040 REM lichaam
1050 PLOT x,123: DRAW 0 ,-20
1055 PLOT x,101: DRAW 0 ,-19
1060 REM benen
1070 PLOT x-15,66: DRAW 15,15: DRAW 15,-15
1080 REM armen
1090 PLOT x-15,117: DRAW 15,-15: DRAW 15,15
1100 RETURN
```

```
2000 DATA 120 ,65,135,0 ,184,65,0 ,91
2010 DATA 168,65,16,16,184,81,16,-16
2020 DATA 184,156,68,0 ,184,140 ,16,16
2030 DATA 204,156,-20 ,-20 ,240 ,156,0 ,-16
```

BIJLAGE E

Binair en hexadecimaal

Deze bijlage beschrijft hoe computers tellen in het binaire systeem.
De meeste Europese talen tellen in een min of meer regelmatig patroon van tientallen. In het Nederlands bijvoorbeeld is het begin wat rommelig, maar al snel wordt het zeer regelmatig:

twintig, eenentwintig, tweeëntwintig, ...negenentwintig
dertig, eenendertig, tweeëndertig, ...negenendertig
veertig, eenenveertig, tweeënveertig, ...negenendertig

Door het gebruik van Arabische cijfers wordt dit alles nog systematischer. De enige reden dat we tot tien tellen is dat we tien vingers aan twee handen hebben.
In plaats van het *decimale* systeem, met tien als basis, gebruiken computers een zestientallig stelsel *hexadecimaal* (of *hex*) genoemd. Omdat we maar 10 cijfers in ons getalstelsel hebben, hebben we 6 cijfers extra nodig om te kunnen tellen. Daarvoor gebruiken we A, B, C, D, E en F. En wat komt er na F? Net als we 10 schrijven voor tien, wordt zestien geschreven als 10. Het getalssysteem begint net als ons tientallig stelsel:

Hex	*Nederlands*
0	nul
1	een
2	twee
:	:
:	:
9	negen

maar gaat daarna anders verder

A	tien
B	elf
C	twaalf
D	dertien
E	veertien
F	vijftien
10	zestien

11	zeventien
:	:
:	:
19	vijfentwintig
1A	zesentwintig
1B	zevenentwintig
:	:
:	:
1F	eenendertig
2Ø	tweeëndertig
21	drieëndertig
:	:
:	:
9E	honderdachtenvijftig
9F	honderdnegenenvijftig
AØ	honderdzestig
A1	honderdeenenzestig
:	:
B4	honderdtachtig
:	:
FE	tweehonderdvierenvijftig
FF	tweehonderdvijfenvijftig
1ØØ	tweehonderdzesenvijftig

Als u hexadecimale notering gebruikt en u wilt dat daar geen verwarring over kan ontstaan, dan moet u telkens achter een getal een 'h' zetten. Bij het spreken zegt u telkens 'heks'. Voor honderdachtenvijftig schrijft u '9EH' en zegt u 'negen E heks'.
Wellicht vraagt u zich af wat dit alles met computer te maken heeft. Computers gebruiken namelijk geen 16 cijfers, maar twee cijfers; Een laag voltage, of uit (Ø), en een hoog voltage, of aan (1). Dit wordt het *binaire* systeem genoemd. De twee binaire cijfers zijn de *bits*. Een bit is dus of Ø of 1.
Als we de verschillende telsystemen naast elkaar zetten, krijgen we:

Nederlands	*Decimaal*	*Hexadecimaal*	*Binair*
nul	Ø	Ø	Ø of ØØØØ
een	1	1	1 of ØØØ1
twee	2	2	1Ø of ØØ1Ø
drie	3	3	11 of ØØ11
vier	4	4	1ØØ of Ø1ØØ
vijf	5	5	1Ø1 of Ø1Ø1
zes	6	6	11Ø of Ø11Ø
zeven	7	7	111 of Ø111
acht	8	8	1ØØØ

negen	9	9	1001
tien	10	A	1010
elf	11	B	1011
twaalf	12	C	1100
dertien	13	D	1101
veertien	14	E	1110
vijftien	15	F	1111
zestien	16	10	10000

Belangrijk hierbij is dat zestien gelijk is aan twee tot de vierde macht, waardoor het omzetten van hex naar binair erg eenvoudig is.
Om hex naar binair om te zetten, moet u elk hex-cijfer omzetten in vier bits, met behulp van de tabel hiervoor.
Om binair naar hex om te zetten, moet u het binaire getal in groepen van vier bits verdelen. Begin rechts en verander elke groep in het bijhorende hex-cijfer.
Omdat dit omzetten zo eenvoudig is, schrijven menselijke wezens de getallen die opgeslagen worden in een computer meestal als hex-cijfers, ook al gebruikt de computer niets anders dan binaire codes.
De bits binnen een computer zijn meestal gegroepeerd in stellen van acht, ook wel *bytes* genoemd. Een enkele byte kan een getal aanduiden tussen nul en 255 (11111111 binair of FF hex), of een karakter uit de ZX Spectrum karakterset. De waarde kan geschreven worden in twee hex-cijfers.
Twee bytes kunnen samengevoegd worden om een zogenaamd 'woord' te vormen. Een woord kan geschreven worden met zestien bits of vier hex-cijfers. Een woord kan een getal aangeven van 0 tot (in decimale notering) 2 tot de macht zestien - 1 = 65535.
Een byte heeft altijd 8 bits, maar woorden kunnen van computer tot computer verschillen.
De **BIN** notatie van hoofdstuk 14 geeft de mogelijkheid om getallen in binaire vorm aan de ZX Spectrum op te geven: **BIN** 0 betekent nul, **BIN** 1 betekent 1, **BIN** 10 betekent 2, etc.
U kunt hiervoor alleen 0-en en 1-en gebruiken. Het getal moet dus een niet-negatief geheel getal zijn. U kunt niet schrijven **BIN** -11 om min drie aan te geven. Daarvoor moet u schrijven -**BIN** 11. Het getal mag niet groter worden dan 65535 (decimaal), want er kunnen niet meer dan 16 bits in.
ATTR is in feite binair. Als u het resultaat van **ATTR** in binair omzet, kunt u dit resultaat in acht bits opschrijven.
De eerste bit is 1 voor knipperend, 0 voor stabiel.
De volgende drie bits zijn de code voor de papierkleur, binair genoteerd.
De laatste drie bits zijn de code voor de inktkleur, binair genoteerd.
De kleurcodes zijn uiteraard ook allemaal binair. Elke code kan in binair codering geschreven worden in drie bits. De eerste voor groen, de tweede voor rood en de derde voor blauw.
Zwart wil zeggen helemaal geen licht, dus alle bits zijn 0 (uit). De code voor zwart is 0000 binair, of nul.

De primaire kleuren groen, rood en blauw hebben alle drie slechts een van de drie bits aan (1). De codes zijn 100, 010 en 001 binair, of vier, twee en een.
De andere kleuren zijn mengsels van deze primaire kleuren, dus hun codes in binair hebben twee of meer bits 1.

Index

Deze index geeft waar nodig aan hoe u een bepaald woord met het toetsenbord kunt invoeren. (De modus -.................en welke SHIFT-toets).
Normaal gesproken wordt elk begrip maar eenmaal per hoofdstuk opgenomen. Heeft u een bepaalde verwijzing gevonden, bekijk dan ook de rest van het hoofdstuk.

K , L of C , SYMBOL SHIFT E.	60
K , L of C , SYMBOL SHIFT W.	59
K , L of C , CAPS SHIFT 5.	
K , L of C , CAPS SHIFT 8.	50
K , L of C , CAPS SHIFT 6.	47
K , L of C , CAPS SHIFT 7.	47

SINCLAIR GEBRUIKER

Het blad dat u diepgaand blijft informeren!

Gefeliciteerd met de aankoop van uw nieuwe Sinclair ZX Spectrum. U deed een prima keuze. Uw Spectrum kent duizend-en-één mogelijkheden. Of het nu gaat om spelletjes, zakelijke programma's of om de lol van het programmeren zélf: u zult zeker heel veel plezier aan uw nieuwe microcomputer kunnen beleven.

Wilt u echter het volle profijt hebben van uw Spectrum, dan is er nog iets meer nodig: informatie. Vrijwel wekelijks verschijnen er nieuwe produkten op de markt waar u als Spectrum-bezitter profijt van kunt hebben. Programma's, randapparatuur, nieuwe spelletjes. Bovendien is er veel méér uit uw Spectrum te halen dan in dit handboek is beschreven. De BASIC-beginselen kent u nu, maar geldt dat ook voor machinetaal of een van de andere computertalen die uw Spectrum kan verwerken?
En natuurlijk hebt u vragen. Hoe sluit ik apparaat A of B aan op m'n Spectrum? Kan ik zélf een spel programmeren? Wat moet ik doen om een programma van een andere computer binnen te halen?

Op deze — en nog vele andere — vragen krijgt u antwoord als u zich abonneert op *Sinclair Gebruiker*. Een blad speciaal voor u. Met informatie over alles wat voor gebruikers van Sinclair-computers van belang kan zijn. Hardware, software, boeken, listings van interessante programma's, een vragenrubriek, advertenties van andere gebruikers en nog veel, veel meer.
Sinclair Gebruiker verschijnt 11 × per jaar en komt dus vrijwel iedere maand bij u in de bus. Een abonnement kost f 59,50 per jaar.
Bovendien hebben we speciaal voor u als kersverse Spectrum-bezitter een **zeer aantrekkelijk introductie-aanbod**! Als u zich nu abonneert, krijgt u de eerste twee nummers gratis. U betaalt voor 11 nummers, maar krijgt er 13!
Een goed idee: word abonnee. Vul een van de twee antwoordkaarten hiernaast in en stuur 'm op. Een postzegel is niet nodig. U ontvangt dan snel het eerstvolgende nummer thuis.

Speciaal introductie-aanbod!

Word nu abonnee en ontvang de eerste twee nummers van *Sinclair Gebruiker* gratis.

Een postzegel
is niet nodig

SINCLAIR GEBRUIKER
Antwoordnummer 1
2300 VB LEIDEN

Een postzegel
is niet nodig

SINCLAIR GEBRUIKER
Antwoordnummer 1
2300 VB LEIDEN

JA, ik word abonnee van Sinclair Gebruiker.
Ik ontvang de eerstvolgende twee nummers gratis.
Ik wil graag een jaarabonnement à f 59,50.
Ik wacht met betalen totdat ik de acceptgirokaart van *Sinclair Gebruiker* heb ontvangen.

Naam: ....

Adres: ....

Postcode/woonplaats: ....

Datum .... Handtekening ....

Ik bezit een ☐ ZX 80 ☐ ZX 81 ☐ ZX Spectrum

Aankoopjaar ☐ '84 ☐ '83 ☐ '82

☐ Ik ben geïnteresseerd in de nieuwe Sinclair QL microcomputer.

Ik gebruik mijn micro voor: ☐ het schrijven van programma's ☐ spelletjes ☐ ....

Ik zou in Sinclair Gebruiker graag de volgende artikelen zien: ....

....

....

JA, ik word abonnee van Sinclair Gebruiker.
Ik ontvang de eerstvolgende twee nummers gratis.
Ik wil graag een jaarabonnement à f 59,50.
Ik wacht met betalen totdat ik de acceptgirokaart van *Sinclair Gebruiker* heb ontvangen.

Naam: ....

Adres: ....

Postcode/woonplaats: ....

Datum .... Handtekening ....

Ik bezit een ☐ ZX 80 ☐ ZX 81 ☐ ZX Spectrum

Aankoopjaar ☐ '84 ☐ '83 ☐ '82

☐ Ik ben geïnteresseerd in de nieuwe Sinclair QL microcomputer.

Ik gebruik mijn micro voor: ☐ het schrijven van programma's ☐ spelletjes ☐ ....

Ik zou in Sinclair Gebruiker graag de volgende artikelen zien: ....

....

....

SINCLAIR ZX SPECTRUM HANDBOEK

Het officiële handboek voor de gebruiker van de Sinclair ZX Spectrum microcomputer. Nu ook in het Nederlands. Onmisbaar als u de mogelijkheden van uw Spectrum ten volle wilt benutten.
Beginners en gevorderden vinden in dit handboek alle noodzakelijke informatie. In een korte, overzichtelijke introductie worden de eerste beginselen behandeld: de aansluiting van t.v. en cassetterecorder, het laden en bewaren van programma's. Vervolgens maakt de lezer stap voor stap kennis met de uitgebreide BASIC van de Spectrum. Ondersteund met duidelijke voorbeeldprogramma's worden alle statements diepgaand besproken. Het aansluiten van randapparatuur (printers, microdrives) wordt kort aangeduid. Ter afsluiting komt de machinetaal van de Spectrum aan de orde, en geven de auteurs een compleet overzicht van de foutmeldingen, de karakterset en de aanwezige BASIC-statements.

ISBN 90-6854-002-5
710 - 30

MICRO PRESS